

B Y T E B O O K S

BUILD YOUR OWN Z80 COMPUTER

Design Guidelines
and
Application Notes



by Steve Ciarcia

Copyrighted material

Build Your Own Z80 Computer

Design Guidelines
and
Application Notes

Steve Ciarcia

Build Your Own Z80 Computer

Copyright © 1981 by Steve Ciarcia. All rights reserved. Printed in the United States of America. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

The author of the circuits and programs provided with this book has carefully reviewed them to ensure their performance in accordance with the specifications described in this book. Neither the author nor BYTE Publications Inc., however, make any warranties concerning the circuits or programs and assume no responsibility or liability of any kind for errors in the circuits or programs, or for the consequences of any such errors. The circuits and programs are the sole property of the author and have been registered with the United States Copyright Office.

The author would like to acknowledge that portions of this book have been reprinted by permission of the manufacturers. The instruction codes in Chapter 3 and Z80 CPU technical information have been reprinted by permission of Zilog, Inc. Chapter 9 is based on an application note reprinted by permission of SMC Microsystems Corporation.

Library of Congress Cataloging in Publication Data

Ciarcia, Steve.

Build your own Z80 computer.

Includes Index.

1. Electronic digital computers—Amateurs' manuals.
2. Zilog Model Z-80 (Computer) I. Title.

TK9769.C52

621.3819382

81-4335

ISBN 0-07-010962-1

AACR2

5 6 7 8 9 10 KGP KGP 8 9 8 7 6 5 4 3

ISBN 0-07-010962-1

Text set in Paladium by BYTE Publications
Edited by Bruce Roberts and Nicholas Bedworth
Design and Production Supervision
by Ellen Klampert
Production by Mike Lonsky
Cover Photo by Charley Freiberg
Copy Edited by Rich Friedman and Peg Clement
Figure and Table Illustrations
by Tech Art Associates
Printed and bound using 45# Bookmark
by Kingsport Press, Kingsport, Tennessee

To my wife Joyce,
Steve Sunderland, and Judy and Lloyd Kishinsky

Build Your Own Z80 Computer

This One



SWP6-P3T-PERL

Introduction

A few years ago, when microprocessors were first introduced, computer enthusiasts and electrical engineers were one and the same. Those of us who lived only to solder kluge after kluge basked in our glory. Now, however, the prices of completely assembled and packaged systems have plummeted. Today anyone with an interest, almost regardless of technical capabilities, can own and operate a computer. Buying a computer is now similar to purchasing a television set and the ranks of computer enthusiasts have swelled accordingly.

With any popular movement, the available literature reflects the concerns of a majority of the followers. And, consistent with the popularization of computer science, the technical emphasis on computer bookshelves has shifted away from hardware design. Other than introductory texts called, say, *How Logic Gates Work*, most computer books either treat microcomputer hardware simplistically or attempt to be "catch-all" cookbooks, sometimes omitting tasty ingredients. Often, the only alternatives are engineering texts and trade journals, tedious reading at best.

For a number of years, I have been writing a column for *BYTE* magazine, and reader response has shown that there still exists a great deal of interest in hardware design and do-it-yourself projects. At the same time, I've been painfully aware of the lack of materials for such people. Most queries come from technical or high school students who have read all the descriptions and studied the block diagrams, but who crave practical answers and system examples. Unfortunately, there are very few books I can suggest.

Build Your Own Z80 Computer is a book written for technically minded individuals who are interested in knowing what is inside a microcomputer. It is for persons who, already possessing a basic understanding of electronics, want to build rather than purchase a computer. It is not an introductory electronics handbook that starts by describing logic gates nor on the other hand is it a text written only for engineering students. While serving to educate the curious, the objective of this book is to present a practical, step-by-step analysis of digital computer architecture, and the construction details of a complete and functional microcomputer.

The computer to be constructed is called a Z80 Applications Processor—ZAP computer for short. It is based on the industry standard Zilog Z80 microprocessor chip. This chip was chosen on the basis of its availability and low cost, as were the other components for ZAP. To further help the homebrew enthusiast, and for those experimenters who prefer to start a book at the back, I have listed in Appendix A a company that supplies programmed EPROMs (erasable-programmable read-only memory).

I have structured the book as a logical sequence of construction milestones interspersed by practical discussions on the theory of operation. My purpose is twofold: to help a potential builder gain confidence, and to make the material more palatable through concrete examples.

Though this is basically a construction manual, considerable effort is given to the "why's" and "how's" of computer design. The reader is exposed to various subjects, including: the internal architectures of selected microprocessors, memory mapping, input/output interfacing, power supplies, peripheral communication, and programming. All discussions try to make the reader aware of each individual component's effect on the total system. Even though I have documented the specific details of the ZAP computer, it is my intention (and the premise of the book) that the reader will be able to configure a custom computer. ZAP is an experimental tool that can be expanded to meet a variety of applications.

ZAP is constructed as a series of subsystems that can be checked and exercised independently. The first item to be built is the power supply. This is a good way to test ability and provide immediate positive reinforcement from successful construction. The three-voltage supply is both overvoltage and overtemperature protected and has adequate current for an expanded ZAP system.

Next, the reader learns why the Z80 was chosen for ZAP and the architectural considerations that affect component selection on the other subsystems. A full chapter is devoted to the Z80 chip. Each control signal is explained in detail and each instruction is carefully documented.

The hardware construction proceeds in stages with intermediate testing in order to ensure success. The basic elements of the computer are assembled first and then checked out. The reader selects which peripherals are to be added. The book contains sections on the construction of a hexadecimal display, keyboard, EPROM programmer, RS-232C serial interface, cassette mass storage system, and fully functional CRT terminal. In addition, a chapter addresses interfacing the ZAP to analog signals. I provide specific circuits that can convert ZAP into a digital speech synthesizer or a data acquisition system and data logger.

A special 1 K (1024 bytes) software monitor coordinates the activities of the basic computer system and the peripherals. Software is explained through flow diagrams and annotated listings. With this monitor as an integral component, ZAP can function as a computer terminal, a dedicated controller, or a software development system.

Build Your Own Z80 Computer is a book for hardware people. It cuts through the theoretical presentations on microcomputers and presents a real "How-to" analysis suitable for the reader with some electronics experience or for the novice who can call someone for supervision. From the power supply to the central processor, this book is written for people who want to understand what they build.

Steve Ciarcia
May 1981

TABLE OF CONTENTS

Introduction

Chapter 1	Power Supply	1
Chapter 2	Central Processor Basics	21
Chapter 3	The Z80 Microprocessor	27
Chapter 4	Build Your Own Computer - Start With the Basics	91
Chapter 5	The Basic Peripherals	129
Chapter 6	The ZAP MONITOR Software	151
Chapter 7	Programming an EPROM	173
Chapter 8	Connecting ZAP to the Real World	183
Chapter 9	Build a CRT Terminal	213

Appendix A Construction Techniques 225

Appendix B ASCII Codes 229

Appendix C Manufacturers' Specification Sheets 233

C1	2708 8K (1K \times 8) UV Erasable PROM	235
C2	2716 16K (2K \times 8) UV Erasable PROM	239
C3	2102A 1K \times 1 Bit Static RAM	243
C4	2114A 1K \times 4 Bit Static RAM	247
C5	8212 8-Bit Input/Output Port	251
C6	KR2376-XX Keyboard Encoder Read-Only Memory	259
C7	COM2017 Universal Asynchronous Receiver Transmitter	263
C8	CRT 5027 CRT Video Timer and Controller	271
C9	CRT 8002 Video Display Attributes Controller	279
C10	COM8046 Baud Rate Generator	287

Appendix D ZAP Operating System 293

Appendix E Z80 CPU Technical Specifications 307

E1	Electrical Specifications	309
E2	CPU Timing	313
E3	Instruction Set Summary	321

Glossary 325

Index 329

CHAPTER 1

POWER SUPPLY

It's not enough to build a central processor card with a little input/output (I/O) and memory, and call it a computer. From the time you walk over to the computer and flip the switch, the system is completely dependent upon the proper operation of its power supply. A book concerned with building a computer system from scratch would be completely inadequate without a description of how to construct an appropriate power supply.

Much has been written on the subject of direct current (DC) power supplies. There are DC to DC and AC (alternating current) to DC converters, switching and shunt regulators, constant voltage transformers, and so on. It's not my intention to make a power supply expert out of everyone. Instead, I will outline the design of the specific DC power supply which we will use to power the Z80 Applications Processor (ZAP).

In large computers, the DC supplies convert enormous amounts of power to run thousands of logic chips; by necessity, manufacturers choose the most efficient methods of power conversion. These state of the art methods would be expensive and difficult for the hobbyist to build in prototype form. Fortunately, the power demands for ZAP are much less than those of the large computers; we can take advantage of established design methods while incorporating the latest advances in regulator technology. Figure 1.1 is a block diagram of the power supply for ZAP.

Each of the three DC supplies necessary to power ZAP consists of three basic modules: a transformer section to reduce the 120 VAC line voltage to the lower voltage used by the computer; an input rectifier/filter to convert AC to low ripple DC; and a regulator which stabilizes the output at a fixed voltage level. Overvoltage protection circuitry will be discussed separately.

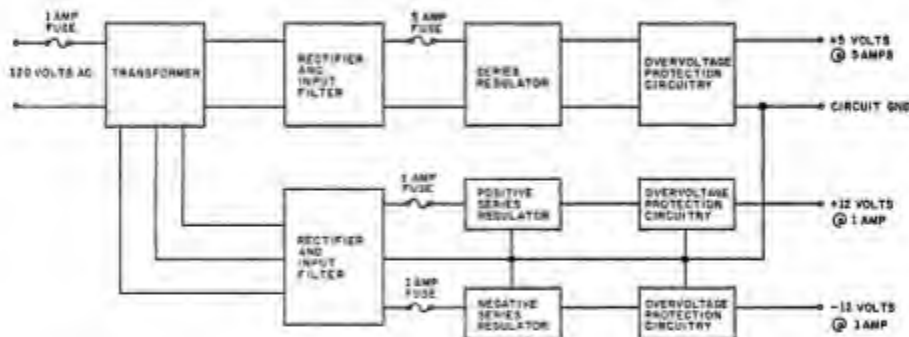


Figure 1.1 A block diagram of the basic power supply for the Z80 Applications Processor (ZAP).

The proper specification of the transformer and input filter is often neglected by hobbyists who overlook the consequences of a poorly designed filter. This is caused, in part, by the abundant technical information circulated by semiconductor manufacturers extolling the virtues of their regulator circuits. One can easily conclude from this "publicity gap" that the regulation section of the power supply is the only component worthy of consideration; and in fact, advances in regulator design and the advent of high-power, three-terminal regulators have reduced the need for the analog designer in the application. In the past, 25-odd components and considerable calculations were necessary to produce an adequate voltage regulator. Now, however, the majority of applications can be accommodated with a single, compact device. Even so, an input filter section should not be taken lightly and still requires thorough consideration and a modest amount of computation for each application.

There are three supply voltages necessary to operate ZAP. Each supply incorporates an input filter section. Because the +5 V supply is the most important, it receives the most attention. For the purposes of this discussion, we will divide the supply into two sections: transformer/input filter, and output regulator.

A standard input filter block diagram is shown in figure 1.2. In its simplest form, it consists of three components that function as follows:

- A transformer that isolates the supply from the power line and reduces the 120 VAC input to usable, low-voltage AC.
- A bridge rectifier that converts AC to full-wave DC and satisfies the charging current demands of the filter capacitor.
- A filter capacitor that maintains a sufficient level between charging cycles to satisfy the regulator input voltage limitations.



Photo 1.1 120 VAC RMS input/output waveform of a saturated transformer.



Photo 1.2 Rectifier waveform.



Photo 1.3 Ripple waveform at various loads.

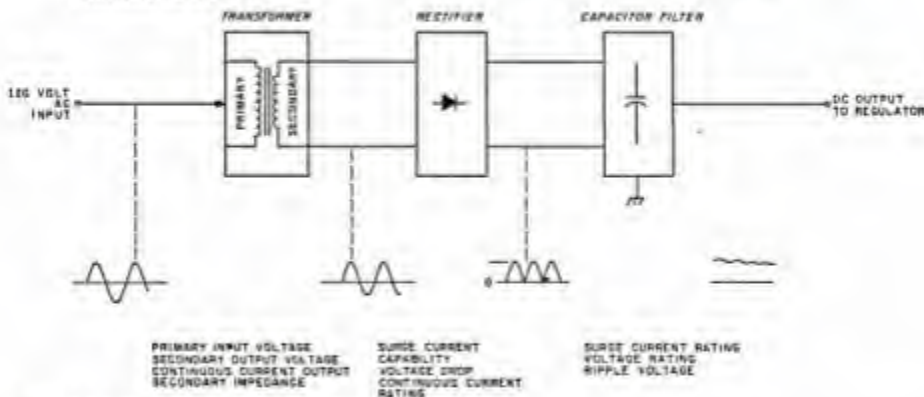


Figure 1.2 A block diagram of a standard input filter.

DESIGNING AN INPUT FILTER

You would think that specifying the transformer would be the first consideration when designing a power supply. Yes and no. The approximate output voltage can be determined by rule of thumb, but the exact requirements are deduced only by a thorough analysis that proceeds from the desired output voltage back. In practice, the difference between a reasonable guess and a laborious analysis will be important only to a person capable of manufacturing his own transformer. In most instances, the hobbyist will have to rely upon readily acquired transformers with standard output voltages. For this reason, my approach is predicated on the practical aspects of power supply design rather than on the minute engineering details that have no real bearing on the outcome.

A 120 VAC RMS (root mean square) sine wave is applied to the primary of the transformer. Figure 1.2 illustrates the waveforms anticipated at selected points through the filter section. Photo 1.1 shows that 120 VAC is actually 340 V peak to peak; care should be used in the insulation and mounting of components.

The secondary output of the transformer will be a similar sine wave, reduced in voltage. It is then applied to a full-wave bridge and the waveform will appear as in photo 1.2. You'll notice a slight flat spot between "humps." As a result of dealing with actual electronic components rather than mathematical models, we should be aware of certain peculiarities. Silicon diodes exhibit threshold characteristics and, in fact, have a voltage drop of approximately 1 V across each diode. This voltage drop becomes significant in full-wave bridge designs and, as figures 1.3a, 1.3b, and photo 1.2 illustrate, can accumulate as diodes are added in series. The 2 V loss in the bridge is an important consideration and should be reflected in the calculations.

The voltage regulator requires a certain minimum DC level to maintain a constant output voltage. Should the applied voltage dip below this point, output stability is

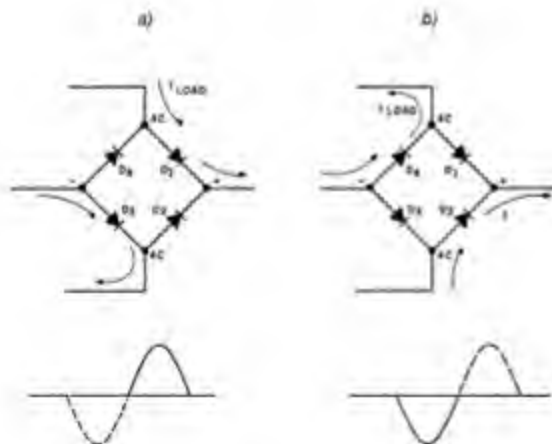


Figure 1.3 The direction of the current flow through the full-wave bridge.

- During the positive half of the AC cycle, current flow is through D_1 and D_4 ; D_2 and D_3 are not conducting. $V_{D1} + V_{D4} = 2$ volts.
- During the negative half of the AC cycle, current flow is through D_2 and D_3 ; D_1 and D_4 are not conducting. $V_{D2} + V_{D3} = 2$ volts.

severely degraded. Thus, a filter capacitor is used to smooth out the “humps” in the rectified sine wave. When the diodes are conducting, the capacitor stores enough charge to maintain the minimum voltage required until the next charge cycle. (In practice, we wouldn’t want to cut it that close.) The input to the transformer is 60 Hz, but because of the characteristics of full-wave rectification, the charging cycles occur at 120 Hz. The capacitor charges up during one 8.3 ms cycle, and, as the regulator draws power from it to satisfy the load demands, it must continue to provide at least the highest minimum input voltage required by the regulator until the next charge cycle, 8.3 ms later. This periodic charge/discharge phenomenon is shown in photo 1.3. The magnitude of the voltage fluctuation between the two peaks of the cycle is referred to as ripple. The highest magnitude of the waveform including the ripple is designated as peak voltage. Both are important to remember and are shown in figure 1.4.

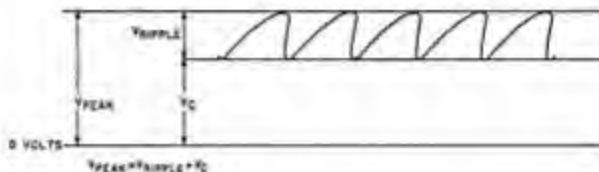


Figure 1.4 Output voltage as a combination of a certain steady-state voltage (V_C) plus a ripple voltage (V_{ripple}).

Given a basic understanding of the individual components at this stage, we can proceed to the case at hand: a 5 V, 5 A power supply. For reasons we’ll discuss later, the 5 V regulator section of this supply will require an absolute minimum of 8.5 V for proper operation. This means that whatever the magnitude of V_{PEAK} and V_{RIPPLE} , the final V_C level must not go below 8.5 V, or the regulator will not work. By giving ourselves some leeway, say $V_C = 10$ V, we can take a little more poetic license with the calculations and still produce a good design. Going much above 10 V, while still satisfying the input criteria, would increase power dissipation and possibly destroy the regulator. There is an answer to this vicious circle and that’s to be conservative. Experience shows that adding a little insurance is worthwhile.

Now that 10 V is the goal, we can appropriately select the other filter components to meet it. Figure 1.5 is the filter circuit of our 5 V supply. R_s is the resistance of the secondary winding of the transformer. For a 5 to 8 A transformer, it will average about 0.1 ohms. The first values to recognize follow:

$$\begin{aligned} V_C &= V_{\text{REGULATOR MINIMUM INPUT VOLTAGE}} = 10 \text{ V} \\ I_{\text{OUT}} &= I_{\text{REGULATOR LOAD}} = 5 \text{ A} \\ R_s &= R_{\text{TRANSFORMER SECONDARY RESISTANCE}} = 0.1 \text{ ohms} \end{aligned}$$

V_{PEAK} can be any voltage up to the maximum input for which the regulator is rated. However, this will increase the circuit power dissipation. The rule of thumb I use when designing supplies of this type is that V_{PEAK} should be approximately 25% higher than V_C . In this way, the capacitor value will be kept within reasonable limits. The ratio of V_C to ($V_{\text{PEAK}} - V_C$) is referred to as the ripple factor of the filter capacitor.

$$Y_F = \frac{V_{\text{PEAK}} - V_C}{V_C} = \frac{12.5 - 10}{10} = 25\%$$

A ripple factor of 25% at 5 A will fall well within the acceptable capacitor ripple current ratings and eliminate the need for the hobbyist to dig into manufacturers’ specifications of capacitors. This ripple factor is arbitrary, but it is best to keep it as low as possible.

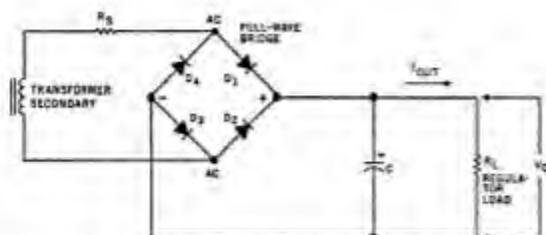


Figure 1.5 The input filter circuit of the 5 V power supply.

SIZING THE CAPACITOR

We now know that the capacitor must sustain 10 V from a peak input of 12.5 V.

$$\left. \begin{array}{l} V_{\text{PEAK}} = 12.5 \text{ V} \\ V_C = 10 \text{ V} \\ V_{\text{RIPPLE}} = 2.5 \text{ V} \end{array} \right\} V_C = V_{\text{PEAK}} - V_{\text{RIPPLE}}$$

The next consideration is to choose a capacitor that will accomplish this goal. Another rule of thumb calculation that saves considerable labor is

$$C = \frac{dt}{dv} \text{ I}$$

where

C = capacitor value in farads = ?

I = maximum regulator current = 5 A

dt = charging time of capacitor = 8.3 ms (120 Hz)

dv = allowable ripple voltage = 2.5 V

Plugging in the values of our circuit,

$$C = \frac{(5)(8.3 \times 10^{-3})}{(2.5)} = 16.6 \times 10^{-3} \text{ farads}$$

or,

$$C = 16,600 \text{ microfarads } (\mu\text{F})$$

Generally available commercial electrolytic capacitors have a tolerance of +50 and -20%. To be on the safe side and to make it easier to find a standard stock component, a value of 20,000 μF is better. The added 3,400 μF reduces the ripple by another 0.4 V and gives us a little "insurance." The only other item to consider with the capacitor is operating voltage. Because the design dictates that V_{PEAK} is 12.5 V, this should be a satisfactory rating. However, experience shows that transformers end up running at higher output voltages than labeled and that 12.5 V at 115 VAC hits 13.6 V when the line voltage goes up to 125 VAC. A capacitor voltage of 15 VDC would appear to satisfy the requirement, but I recommend using the next increased standard value of 20 VDC.

The capacitor is therefore 20,000 μF at 20 VDC. The rectifier can be a monolithic full-wave bridge, or it can be four discrete diodes. Note that because a bridge is usually encapsulated, the four terminals are labeled instead of showing the polarity markings of the individual diodes. The designations for the four terminals are two AC input terminals, and a + and - output terminal.

THE RECTIFIER

There are three considerations when choosing a rectifier: surge current rating, continuous current, and PIV (peak inverse voltage) rating. These choices are not inconsequential and must be considered carefully.

When a power supply is first turned on, the capacitor is totally discharged. In fact, it will instantaneously appear to be a 0 ohm impedance to the voltage source. The only aspect of the circuit that limits the initial current flow is the resistance of the secondary transformer windings and the connecting wiring; designers often add a series resistance to limit surge current.

The surge current in this circuit is

$$I_{\text{surge}} = \frac{V_{\text{peak}}}{R_s} = \frac{12.5}{0.1} = 125 \text{ A}$$

and the time constant of the capacitor is

$$\tau = R_s \times C = (0.1)(20 \times 10^{-3}) = 2 \text{ ms}$$

As a rule of thumb, the surge current will cause no damage to the diode if I_{surge} is less than the surge current rating of the diode and if

$$\tau < 8.3 \text{ ms (which it is)}$$

We can't check surge rating until after we choose a diode bridge, but the other two parameters can be defined.

The bridge can be either of the following:

Motorola MDA 980-2: $I_{\text{cont}} = 12 \text{ A}$, $I_{\text{surge}} = 300 \text{ A}$, PIV = 100 V

Motorola MDA 990-2: $I_{\text{cont}} = 27 \text{ A}$, $I_{\text{surge}} = 300 \text{ A}$, PIV = 100 V

Both of the above bridges have a surge current rating of 300 A, so our surge requirement is also satisfied.

PIV

PIV (peak inverse voltage) is the maximum voltage that may appear across the diode before it self-destructs. Diodes, unlike capacitors, are unforgiving; transients will wipe them out. It is not unusual to have 400 V transients on the 115 VAC input line. This causes our 12.5 V to shoot up momentarily to 43 V! The bridge rectifier should therefore have a minimum PIV rating of 50 V. For a few pennies more, you can get a bridge rated for 100 PIV. Remember, insurance costs less than computers.

CONTINUOUS CURRENT

The last consideration is continuous current rating. Whereas the regulator may be designed for a 5 A output, the particular regulator I have chosen will draw 7 A if shorted. This is not standard operating procedure, but it can happen. The suggested standard component would be a 12 A, 50 PIV bridge. A preferred component would be one rated for 12 A at 100 PIV or, for an additional 15% cost premium, a 27 A at 100 PIV. This last design choice is strictly brute force, but it saves the diode bridge should the capacitor ever short-out accidentally. A 6 A transformer might put out more than 12 A in a short-circuit mode, but it's unlikely that it would be capable of 27 A. Either choice will satisfy the design, but only one saves the design from the builder.

THE TRANSFORMER

Now let's consider the transformer. We have determined the voltage drops across the various components. The values are used to calculate the required RMS (root mean

square) secondary voltage in the following way:

$$V_{SEC(RMS)} = \frac{V_C + V_{RIPPLE} + V_{RECT}}{\sqrt{2}} \quad V_{RECT} = \text{Voltage drop across each diode— (approximately 1 V per diode)}$$

$$= \frac{10 + 2.5 + 2.0}{1.414}$$

$$= 10.25 \text{ V}$$

In practice, a 10 V, 6 A standard value transformer will be close enough.

The components of the + and -12 V supplies are chosen in a similar manner, with the exception that required current is only 1 A, and a 200 PIV bridge is recommended because of the particular rectifier configuration. The finished schematic of the transformer and filter section of our computer is illustrated in figure 1.6.

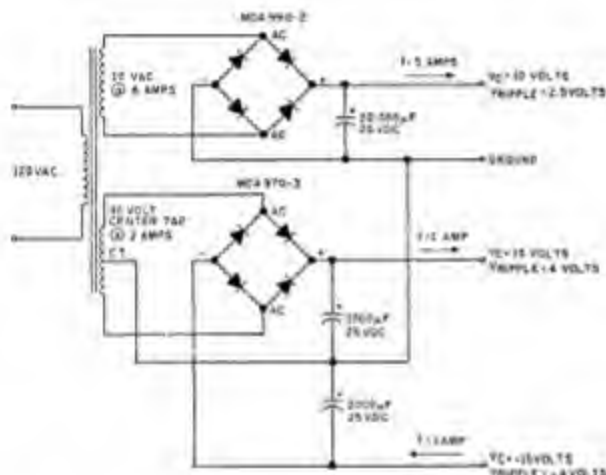


Figure 1.6 A schematic diagram of a transformer and input filter section.

VOLTAGE REGULATORS

The voltage regulator section of our power supply is the next consideration. All voltage regulators perform the same task: they convert a given DC input voltage into a specific, stable DC output voltage and maintain this setpoint over wide variations of input voltage and output load. The typical voltage regulator, as shown in figure 1.7, consists of the following:

- a reference element that provides a known stable reference voltage
- a voltage translation element that samples the output voltage level
- a comparator element that compares the reference and output level to produce an error signal
- a control element that can utilize this error signal to provide translation of the input voltage to produce the desired output

The control element depends on the design of the regulator and varies widely. The control determines the classification of the voltage regulator: series, shunt, or switch-

ing. For the series regulator, the control element regulates the output voltage by modulating the series element, usually a transistor, and causes it to act as a variable resistor (figure 1.8). As the input voltage increases, the series resistance R_s also increases, causing a larger voltage drop across it. In this way, the output voltage (V_{OUT}) is maintained at a constant level.

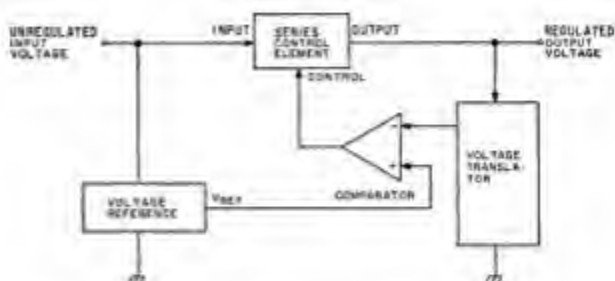


Figure 1.7 A block diagram of a typical voltage regulator.

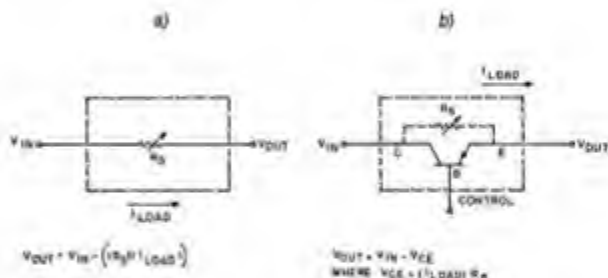


Figure 1.8 A series control element in the voltage regulator.
a) The series control element acts as a variable resistance, R_s .
b) The series element is most often a transistor.

To accomplish this closed-loop control, a reference comparison and feedback system is incorporated into the hardware. A fixed and stabilized reference voltage is easily produced by a zener diode. The current produced is low, however; the device could not serve as a power regulator by itself.

The voltage translator connected to the output of the series control element produces a feedback signal that is proportional to the output voltage. In its simplest form, the voltage translator is a resistor-divider network. The two signals, reference and feedback, provide the necessary information to the voltage comparator for closed loop feedback to occur (figure 1.9). The output of the comparator effectively drives the base of the series pass transistor so that the voltage drop across the transistor will be maintained at a stabilized preset value when subtracted from the input voltage.

Modern power supply designers can still use individual components to construct the modular elements of a series voltage regulator, but most reserve this laborious endeavor for specialized applications. The ZAP computer system outlined here requires +5 V, +12 V, and -12 V. The combined temperature, stability, and drift

tolerances cannot exceed $\pm 5\%$ on any of the three set points. The easiest way to minimize risk is to reduce the number of circuit components to the bare minimum. Other designers had the same idea and thus the three-terminal regulator was invented. Figure 1.10 is the block diagram of such a device.

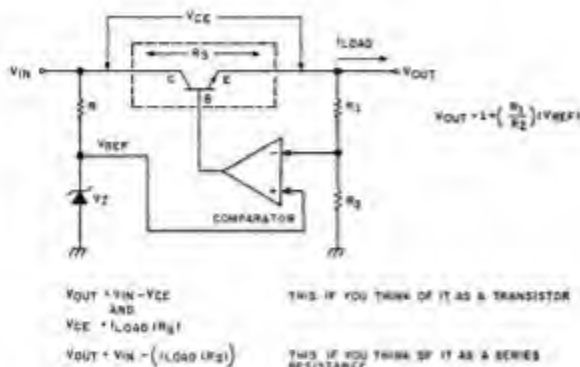


Figure 1.9 A schematic diagram of a series voltage regulator.

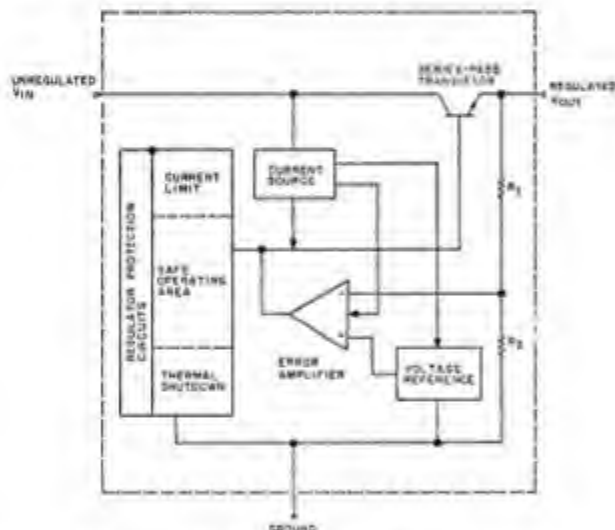


Figure 1.10 A block diagram of a three-terminal voltage regulator.

Basically, a three-terminal regulator incorporates all the individual transistors, resistors, and diodes into a single integrated circuit. While simple to use, these devices have a far more complicated internal structure than the series regulator of figure 1.9. Only three terminals are necessary in applications where the fixed output is a standard value such as: $\pm 5\text{ V}$, $\pm 6\text{ V}$, $\pm 8\text{ V}$, $\pm 12\text{ V}$, $\pm 15\text{ V}$ or $\pm 24\text{ V}$. The three connections are unregulated DC from our input filter, a ground reference, and finally, regulated DC output.

In a three-terminal regulator, the voltage reference is the most important part because any abnormality or perturbation will be reflected in the output. Therefore, the reference must be stable and free from noise or drift. More advanced designs use band-gap reference circuits rather than zener diodes. Because of its complexity, such an approach is practical only in the integrated circuit (IC) environment. Essentially, a band-gap reference voltage is derived from the predictable temperature, current, and voltage relationships of a transistor base-emitter junction.

Another advantage of the three-terminal regulator is that in monolithic circuits, stable current sources can easily be realized by taking advantage of the good matching and tracking capability of monolithic components. Also, as in the previous case, the designer can add as many active devices as necessary without significantly increasing the IC circuit area. Operation of the reference circuit at a constant current level reduces fluctuations due to line-voltage variation. Thus, the output has increased stability. The error amplifier is also operated at a constant current to reduce line-voltage influence.

The most important consideration for the hobbyist is that these chips incorporate protective circuitry, guarding the regulator from certain types of overloads. They protect the regulator against short-circuit conditions (current limit); excessive input/output differential condition (safe operating area); and excessive junction temperatures (thermal limit). Of course, all this circuitry is designed to protect the regulator, not the computer.

CHOOSING A REGULATOR

The 5 A μ A78H05 hybrid voltage regulator has all the inherent characteristics of the monolithic three-terminal regulator (i.e. full protective circuitry). Each hermetically-sealed TO-3 package contains a μ A78M05 monolithic regulator chip driving a discrete series-pass transistor Q1 and two short-circuit-detection transistors Q2 and Q3 (see figure 1.11). The pass transistor is mounted on the same beryllium oxide substrate as the regulator chip, thus insuring nearly ideal thermal transfer between Q1 and the temperature-sensing circuit of the 78M05.

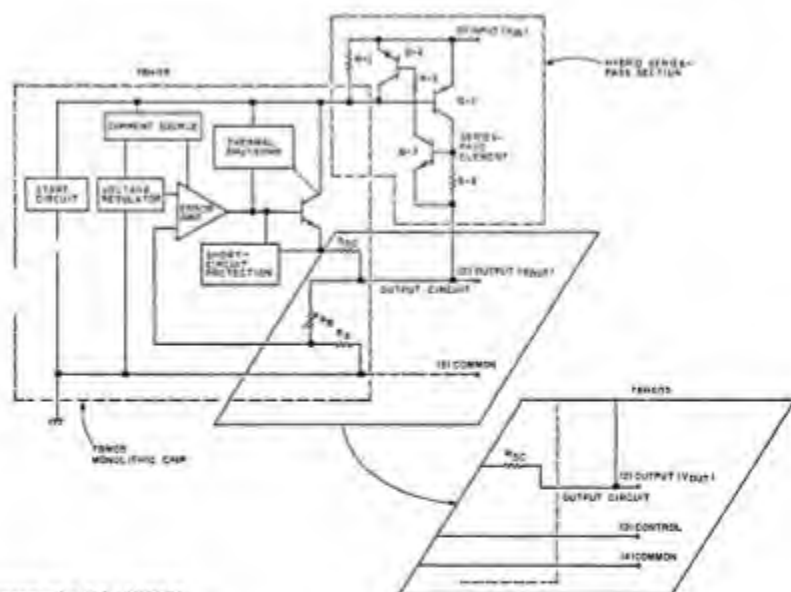


Figure 1.11 A block diagram of a 5 A μ A78H05 and μ A78HG05 hybrid voltage regulator.

ELECTRICAL CHARACTERISTICS: $T_A = 25^\circ\text{C}$, $I_{OUT} = 2.0\text{ A}$ unless otherwise specified.

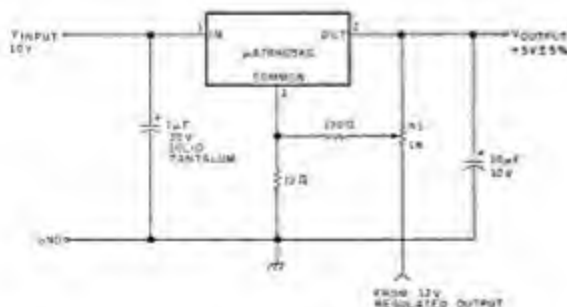
CHARACTERISTICS	CONDITIONS	$\mu\text{A}78\text{H}05$			UNITS
		MIN	TYP	MAX	
Output Voltage	$I_{OUT} = 2.0\text{ A}$, $V_{IN} = 10\text{ V}$	4.8	5.0	5.2	V
Line Regulation	$V_{IN} = 8.5$ to 25 V		10	50	mV
Load Regulation	$10\text{ mA} \leq I_{OUT} \leq 3.0\text{ A}$, $V_{IN} = 10\text{ V}$		10	50	mV
Quiescent Current	$I_{OUT} = 0$, $V_{IN} = V_{OUT} = 5.0\text{ V}$			10	mA
Reverse Protection	$I_{OUT} = 1.0\text{ A}$, $r = 210\text{ }\Omega$, 5.0 V P-P	50			dB
Output Noise	$10\text{ Hz} < f < 100\text{ kHz}$, $V_{IN} = V_{OUT} = 5.0\text{ V}$		40		μVrms
Dropout Voltage	$I_O = 3.0\text{ A}$		3.0		V
	$I_O = 2.0\text{ A}$		2.6		V
Short-Circuit Current Limit	$V_{IN} = 10\text{ V}$		3.0		A

Figure 1.12 Electrical characteristics of the $\mu\text{A}78\text{H}05$ voltage regulator.

The output circuit is designed so that the worst-case current requirement of the Q1 base, added to the current through R2, always remains below the current-limit threshold of the 78H05. Resistor R1, in conjunction with Q2 and Q3, makes up a current sense and limit circuit to protect the series-pass device from excessive current drain.

Safe area protection is achieved by brute force and is designed with the hobbyist in mind. The series-pass transistor is capable of handling the short-circuit current at the maximum input voltage rating of the 78H05. (See figure 1.12 for the electrical characteristics of the 78H05.)

The output of the device is nominally 5.0 V but can vary between 4.8 and 5.2 V. Even though this falls within the $5.0\text{ V} \pm 15\%$ tolerance necessary to run the computer, there might be a problem with the voltage drop in the cabling between the power supply and the computer. Up to 0.5 V could be lost in the wiring and connectors. Remember that at 5 A, a resistance of only 0.1 ohms can cause a 0.5 V drop. Unfortunately, the 78H05 is a fixed-output device when referenced to ground. If 4.8 V happens to come out, "that's all you gets" (sic). But, in a classic case of engineering razzle-dazzle, we can fool the regulator by making the ground reference adjustable. Figure 1.13 shows the circuit that makes this possible. A potentiometer sourced from the -12 V supply creates a relative-ground reference for the 78H05. If the particular device in question had an output of 4.95 V, and we adjusted R1 for a potential of 0.20 V on the common regulator pin, the output referenced to ground would change to $4.95 + 0.20$, or 5.15 V. For the fanatics in the crowd, this particular circuit also allows a high-output device to be reduced to 5.00 V by selecting an appropriate negative voltage ground reference pin.

Figure 1.13 Adding "trim adjust" to the $\mu\text{A}78\text{H}05$ three-terminal voltage regulator.

With the 5 V supply complete, our next concern is the +12 V and -12 V supplies. Other devices within the 7800 family of regulators will satisfy the requirements. The 7812 and a 7912 are 1 A positive and negative regulators respectively; they exhibit the same protection characteristics as the 78H05. Figures 1.14 and 1.15 outline the exact specifications. Because we are dealing with much lower currents than the +5 V supply, there is considerably less concern over voltage losses through connecting cables, and it is unnecessary to add trim adjustment circuitry. Figure 1.16 is the finished schematic of the ZAP power supply. Additional regulator circuit diagrams (figures 1.17a, b, c and d) are included to demonstrate how the 7800 series of regulators can be used in our application. Are we finished yet? Of course not. Close examination of figure 1.16 shows two items not discussed previously: heat sinks and overvoltage protection. These two subjects and a short discussion of the importance of correct layout complete the chapter.

μA7812

ELECTRICAL CHARACTERISTICS: $V_{IN} = +18\text{ V}$, $I_{OUT} = 500\text{ mA}$, $-55^{\circ}\text{C} < T_J < 150^{\circ}\text{C}$, $C_{IN} = 0.33\text{ }\mu\text{F}$, $C_{OUT} = 0.1\text{ }\mu\text{F}$, unless otherwise specified.

CHARACTERISTICS	CONDITIONS	MIN	TYP	MAX	UNITS
Output Voltage	$T_J = 25^{\circ}\text{C}$	-11.5	-12.0	-12.5	V
Line Regulation	$T_J = 25^{\circ}\text{C}$ $-14.5\text{ V} < V_{IN} < 30\text{ V}$		10	120	mV
	$-18\text{ V} < V_{IN} < 22\text{ V}$		5.0	80	mV
Load Regulation	$T_J = 25^{\circ}\text{C}$ $5\text{ mA} < I_{OUT} < 1.5\text{ A}$		12	120	mV
	$250\text{ mA} < I_{OUT} < 500\text{ mA}$		4.0	80	mV
Output Voltage	$-15.5\text{ V} < V_{IN} < 21\text{ V}$ $5\text{ mA} < I_{OUT} < 1.5\text{ A}$ $P < 15\text{ W}$	-11.4		-12.4	V
Quiescent Current	$T_J = 25^{\circ}\text{C}$		4.3	6.0	mA
Quiescent Current Change	with I_{IN}			0.8	mA
	with load			0.6	mA
Output Noise Voltage	$T_A = 25^{\circ}\text{C}$, 10 Hz to 1 kHz		9	40	mV/V _{OUT}
Ripple Rejection	$f = 20\text{ Hz}$, $-15\text{ V} < V_{IN} < 25\text{ V}$	81	71		dB
Dropout Voltage	$I_{OUT} = 1.0\text{ A}$, $T_J = 25^{\circ}\text{C}$		9.0	2.8	V
Output Resistance	$r = 1\text{ }\Omega$		16		mΩ
Short Circuit Current	$T_J = 25^{\circ}\text{C}$, $V_{IN} = 35\text{ V}$		0.76	1.3	A
Peak Current	$T_J = 25^{\circ}\text{C}$	1.2	2.2	3.1	A
Average Temperature Coefficient of Output Voltage	$I_{OUT} = 5\text{ mA}$ $-55^{\circ}\text{C} < T_J < 150^{\circ}\text{C}$			0.4	mV/°C
	$100^{\circ}\text{C} < T_J < 150^{\circ}\text{C}$			0.3	mV/°C

Figure 1.14 Electrical characteristics of the μA7812 voltage regulator.

μA7912

ELECTRICAL CHARACTERISTICS: $V_{IN} = -18\text{ V}$, $I_{OUT} = 500\text{ mA}$, $C_{IN} = 0.33\text{ }\mu\text{F}$, $C_{OUT} = 0.1\text{ }\mu\text{F}$, $-55^{\circ}\text{C} < T_J < 150^{\circ}\text{C}$, unless otherwise specified.

CHARACTERISTICS	CONDITIONS	MIN	TYP	MAX	UNITS
Output Voltage	$T_J = 25^{\circ}\text{C}$	-11.5	-12.0	-12.5	V
Line Regulation	$T_J = 25^{\circ}\text{C}$ $-14.5\text{ V} < V_{IN} < -30\text{ V}$		10	120	mV
	$-18\text{ V} < V_{IN} < -22\text{ V}$		5.0	80	mV
Load Regulation	$T_J = 25^{\circ}\text{C}$ $5\text{ mA} < I_{OUT} < 1.5\text{ A}$		12	120	mV
	$250\text{ mA} < I_{OUT} < 500\text{ mA}$		4.0	80	mV
Output Voltage	$-15.5\text{ V} < V_{IN} < -21\text{ V}$ $5\text{ mA} < I_{OUT} < 1.5\text{ A}$ $P < 15\text{ W}$	-11.4		-12.4	V
Quiescent Current	$T_J = 25^{\circ}\text{C}$		1.5	3.0	mA
Quiescent Current Change	with I_{IN}			1.0	mA
	with load			0.8	mA
Output Noise Voltage	$T_A = 25^{\circ}\text{C}$, 10 Hz to 1 kHz		9	40	mV/V _{OUT}
Ripple Rejection	$f = 20\text{ Hz}$, $-15\text{ V} < V_{IN} < -25\text{ V}$	84	80		dB
Dropout Voltage	$I_{OUT} = 1.0\text{ A}$, $T_J = 25^{\circ}\text{C}$		1.1	2.3	V
Peak Output Current	$T_J = 25^{\circ}\text{C}$	1.3	2.1	3.1	A
Average Temperature Coefficient of Output Voltage	$I_{OUT} = 5\text{ mA}$, $-55^{\circ}\text{C} < T_J < 150^{\circ}\text{C}$			0.3	mV/°C
	$100^{\circ}\text{C} < T_J < 150^{\circ}\text{C}$			0.2	mV/°C
Short Circuit Current	$V_{IN} = -35\text{ V}$, $T_J = 25^{\circ}\text{C}$			1.2	A

Figure 1.15 Electrical characteristics of the μA7912 voltage regulator.

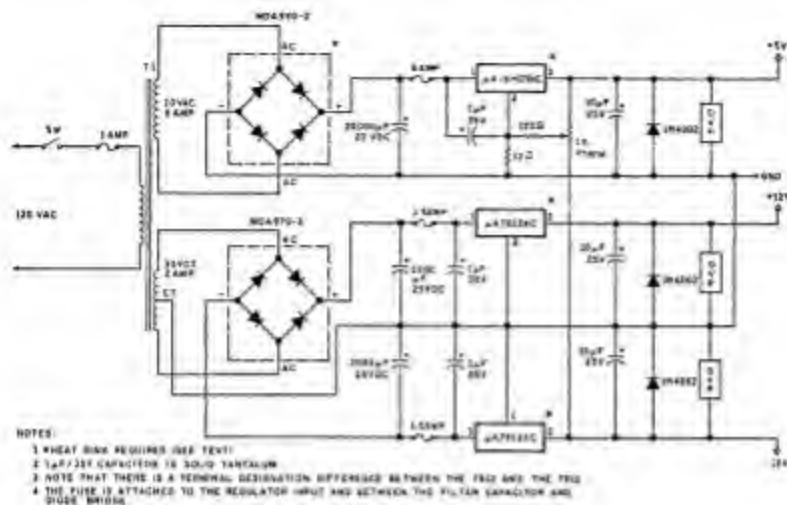


Figure 1.16 A schematic diagram of the finished power supply for the ZAP computer.

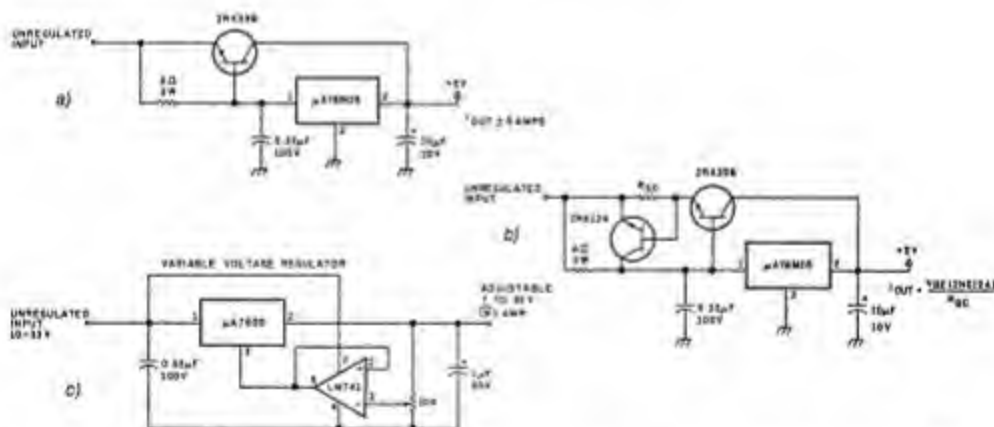
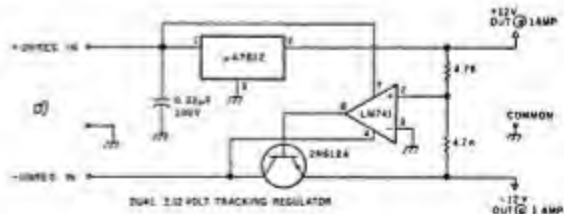


Figure 1.17 Additional voltage regulator circuit diagrams to demonstrate how the 7800 series of regulators can be used.

- A high-current voltage regulator using a 500 mA 78M05 three-terminal regulator.
- A high-current short-circuit protected voltage regulator, an enhanced version of figure 1.17a.
- Using a 7800 + 5 V voltage regulator to produce a higher output voltage.
- A dual ± 12 V tracking voltage regulator.



LAYOUT IS IMPORTANT

Integrated circuit regulators employ wide-band transistors in their construction to optimize response. As a result, they must be properly compensated to ensure stable closed-loop operation. Their compensation can be upset by stray capacitance and line inductance of an improper layout. Circuit lead lengths should be held to a minimum, and external bypass capacitors in particular should be located as close as possible to the regulator control circuit.

Figure 1.18a illustrates a typical layout of the components of our supply, and figure 1.18b details the areas that can cause problems. Improper placement of the input capacitor can induce unwanted ripple on the output voltage. This occurs when the current flowing in the input circuit influences the common ground line of the regulator. The voltage drop produced across R_2' will cause the output of the regulator to fluctuate in the same manner as the voltage trim circuit we discussed previously. The peak currents in the input circuit (which consists of the rectifier and filter capacitor) can be tens of amperes during charge cycles. These high-current spikes can cause substantial voltage drops on long-lead lengths or thin-wire connections. They can also degrade performance to the point that proper input voltage to the regulator cannot be maintained except during low-current operation.

The output current loop is also susceptible to circuit layout. In a three-terminal regulator, the fixed-output voltage $V_{OUT(REG)}$ is referenced between "out" and "common" of the chip. Because the load current flows through R_2' , R_3' , and R_4' , as well as the load itself, these combined voltage losses may reduce V_{OUT} to an intolerable level. Notice that the ground for this circuit is at point C while the present R load is between points A and B. If another load, more memory for example, is connected to this supply between points A and C, it would have a different V_{OUT} . Adjusting the trim setting of such a seesaw supply can be dangerous; it's possible to have one load completely within tolerance and another over or under voltage. One last point to consider is that R_4' serves to negate the purpose of the regulator because it continually reduces V_{OUT} as the load current increases.

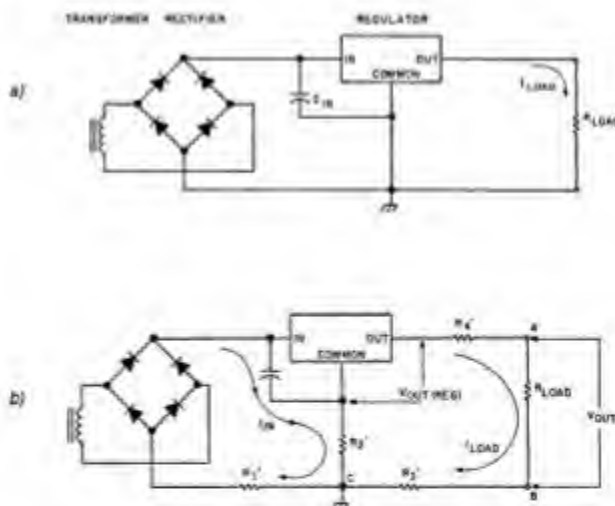


Figure 1.18 A typical layout of the power supply components and associated problems.

a) A typical layout.

b) Errors contributed by the layout in figure 1.18a.

Figure 1.19 is the block diagram of a proper layout. All high-current paths should use heavy wire to minimize resistance and resultant voltage drops. You'll notice now that the input and output circuit current paths are separated effectively. Note that the wires from the rectifier go directly to the capacitor and that two wires from the capacitor send power to the rest of the circuit. If you follow this convention and use two separate pairs of leads, you can eliminate input-circuit induced errors.

Finally, we need to discuss the concept of the single-point ground. One point in the power supply must be designated as ground; the grounds of all other supplies and loads are connected to it. In practical terms, the best way to implement this ground connection is to use a metal strip or several lengths of heavy wire soldered together. The strip is a ground bus with such a low resistance that a voltage measured between point A and any place along the bus will be virtually undetectable. Another +5 V bus should be connected to the output of the supply so that voltage distribution throughout the circuit is consistent. Use thick wire in power supplies. Even if zero-resistance wire isn't easily obtainable, always remember—there is no such thing as wire that is too thick!

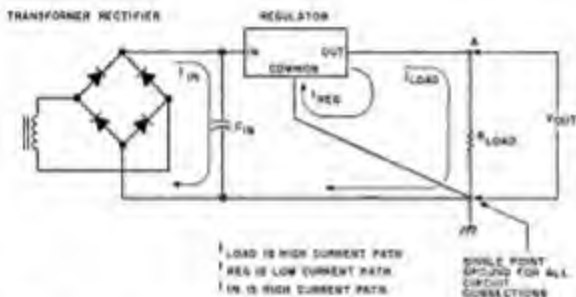


Figure 1.19 A block diagram of a proper layout for the power supply components.

THERMAL CONSIDERATIONS

You've just built the power supply I've outlined, flipped on the power, and everything works. After a few minutes, something happens and the computer suddenly stops running. Naturally, you start looking around and touching things. Eventually, your fingers will end up on the regulator chip. Immediately you scream, jump back, and in the process knock over the computer and your celebration martini. If you are lucky, your fingers will be the only thing burned!

When not properly cooled, the regulators will protect themselves from destruction by reducing their output or completely shutting off. In this case, the system could cease to function. A more catastrophic problem arises from ICs that use all three voltages for normal operation. Loss of one or more of these voltages could permanently damage the device. This will never happen if power dissipation is limited and the proper cooling methods are employed.

The first step is to check the power dissipation of our design with the ratings of the particular devices. In practical terms, power, expressed in watts, is volts times amperes:

$$P_D = E \times I$$

In our 5 V regulator we have $V_C = 10$ V and $V_{REG} = 12.5$ V at 5 A.

$$\begin{aligned}P_{D(\text{RMS})} &= (V_C - V_{\text{OUT}}) \times 5 \text{ A} \\&= (10 - 5) \times 5 \\&= 25 \text{ W}\end{aligned}$$

$$\begin{aligned}P_{D(\text{PEAK})} &= (V_{\text{PEAK}} - V_{\text{OUT}}) \times 5 \text{ A} \\&= (12.5 - 5) \times 5 \\&= 37.5 \text{ W}\end{aligned}$$

$$P_{D(\text{AVERAGE})} = \frac{37.5 + 25}{2} = 31.25 \text{ W}$$

This means that under full load conditions, about 30 W of heat will be produced by the 78H05. The device is fortunately rated for 50 W at 25°C and is still capable of handling 30 W up to 75°C.

Although the internal power dissipation is limited, the junction temperature must be kept below the maximum specified temperature (125°C) in order for the device to function at all. To calculate the heat sink required, there are specific equations to solve.

The required thermal data and calculations follow:

$$\begin{array}{ll}\text{Typical } \theta_{JC} = 2.0 & \text{Maximum } \theta_{JC} = 2.5 \\ \text{Typical } \theta_{JA} = 32 & \text{Maximum } \theta_{JA} = 38 \\ P_{D(\text{MAX})} = \frac{T_{J(\text{MAX})} - T_A}{\theta_{JC} + \theta_{CA}} & \text{for } \theta_{CA} = \theta_{CS} + \theta_{SA}\end{array}$$

Solving for T_J ,

$$T_J = T_A + P_D(\theta_{JC} + \theta_{CA})$$

or without a heat sink,

$$P_{D(\text{MAX})} = \frac{T_{J(\text{MAX})} - T_A}{\theta_{JA}}$$

$$T_J = T_A + P_D \theta_{JA}$$

where T_J = junction temperature

T_A = ambient temperature

P_D = power dissipation

θ_{JC} = junction to case thermal resistance

θ_{JA} = junction to ambient thermal resistance

θ_{CA} = case to ambient thermal resistance

θ_{CS} = case to heat sink thermal resistance

θ_{SA} = heat sink to ambient thermal resistance

$$\theta_{JA} = \frac{T_J - T_A}{P_D} = \frac{125^\circ\text{C} - 25^\circ\text{C}}{31.25 \text{ W}} = 3.2^\circ\text{C/W}$$

Because θ_{JA} as calculated is less than θ_{JA} from the specification sheet, a heat sink is definitely required, and a TO-3 type heat sink of 3.2°C/W is the minimum desired.

Before you size a heat sink for the 78H05, realize that there are two more regulators and two bridge rectifiers that will need heat sinking. Each 12 V regulator will average about 5 W dissipation. The diode bridge associated with the +5 V supply (remember the 2 V drop) dissipates about 10 W while the other is good for 2 W. Therefore, any heat sinks in the power supply must handle more than 50 W.

WHAT IS THE PRACTICAL METHOD FOR CHOOSING HEAT SINKS?

Choosing a heat sink can be easy or hard depending upon your outlook on rule of

thumb measures. We already know that we need a 50 W heat sink. It's easy to assume that buying one "rated for 50 W" from a local electronics supply will solve the problem. What this rating usually means, however, is that if 50 W is applied through a transistor to this sink, and the ambient temperature is 25°C, the surface temperature of the sink will climb to 100°C. Fried eggs anyone?

We must not forget that manufacturers' specs always refer to limiting maximum junction temperature, not to keeping the case cool enough to touch. Personally, I hate red-hot power supplies. To get a heat sink that would take our 50 W and stay about 60-70°C would probably mean getting one rated for 200-300 W! Remember that heat sinks are expensive—and big.

The simplest solution is best. I prefer forced air cooling. Put the 50 W on an economical heat sink of, say, a 100 W rating and put your money into a good fan. You can still run through all the calculations and determine how many square inches you need, but the effect of blowing a little air over a heat sink multiplies its capabilities enormously.

OVERVOLTAGE PROTECTION

The final area to be addressed in the power supply is overvoltage protection. As designed by manufacturers, regulators protect themselves by reducing output voltage or complete shutoff. The chances of computer component damage from low voltage is minuscule by comparison to overvoltage. It is unlikely to happen, but if the 78H05 were to accidentally short out, as much as 12.5 V would be applied to the +5 V bus. You could then kiss the computer good-bye!

+5 volt OVP				12 volt OVP			
D ₁	5.6 V	1N4734		D ₁	13 V	1N4743	
SCR ₁	50 V 25 A	2N682		SCR ₁	50 V 8 A	2N4441	
Fuse	6 amp	fast-blow		Fuse	1.5 amp	fast-blow	

The semiconductor components of this 12 volt OVP are reversed in polarity for the -12 volt OVP.

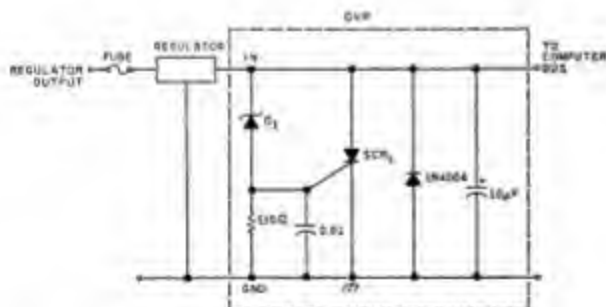


Figure 1.20 A simple overvoltage protection circuit.

The circuit of figure 1.20 is a simple OVP (over-voltage protector). It can be used as shown on the 5 V and 12 V supplies. The appropriate components are listed in the tables of figure 1.20. You'll notice that the fuses are rated higher than the output we've previously discussed. The fuse is for the OVP and not to protect the regulators. Unfortunately, the nature of fast-blow fuses is not to pass 5 A, if it is a 5 A fuse, but to open at 5 A. The fuse must have a higher rating in order to allow circuit operation at 5 A.

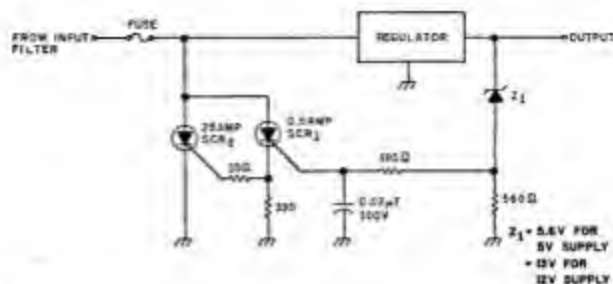


Figure 1.21 A schematic diagram of a more complex overvoltage protection circuit. The crowbar section of the OVP can be located next to the fuse while the OVP sensor Z_1 is located at the regulator output. This is a preferred placement of the parts if the sensor and clamp can be adequately separated. Low-current sensor Z_1 fires SCR_1 in an overvoltage condition. SCR_1 in turn fires high-current SCR_2 . The combination of SCRs allows considerable leeway in the choice of SCR, since the question of gate current becomes less relevant.

Because the short-circuit current of the 78H05 is 7 A, the 25 A silicon-controlled rectifier (SCR) will certainly make short work of the fuse if it triggers. Figures 1.21 and 1.22 are slightly more complex OVP circuits and can also be used.

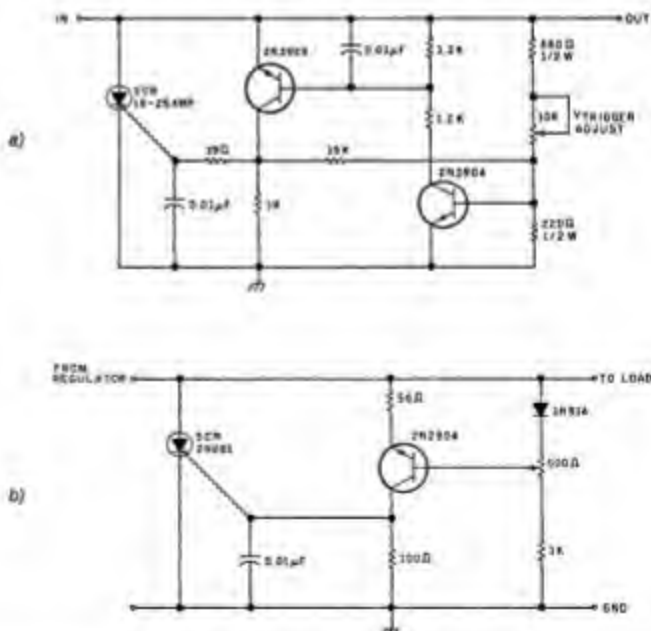


Figure 1.22 Schematic diagrams of adjustable-voltage overvoltage protection circuits.
 a) An adjustable-voltage OVP circuit with an internal current amplifier to drive the SCR gate.
 b) An alternate circuit for a simple adjustable-voltage OVP circuit.

What does an OVP (often called an "overvoltage crowbar") do? It monitors a particular bus voltage and shuts it down if it goes above a predetermined level. OVP circuits can be designed to trigger 1 mV above our 5% tolerance band. Such circuits are not only complicated, but they may also create additional problems through accidental triggerings. The failure modes that are most likely to occur concern a regulator short or accidentally tying two buses together, for example the +5 V and +12 V. In either case, the result is a rapid voltage rise on the output lines. As voltage rises above the zener value, current flows into the SCR gate. At a certain point, usually below where any components would have been damaged, the SCR fires and shorts the output line to ground. The excessive current blows the fuse, eliminating the problem regulator or regulators (both fuses would blow if the +5 V and +12 V were connected). All this occurs very fast. The test circuit of figure 1.23 demonstrates what happens when the +5 V OVP suddenly has +12 V applied. Test circuits are the only way you ever want to see the action of an OVP. If your power supply functions properly, it should never trigger. The SCR never allows the line to go to 12 V before clamping it to ground. Replacing the fuse with a 220 ohm resistor allows multiple applications of the push button without replacing fuses.

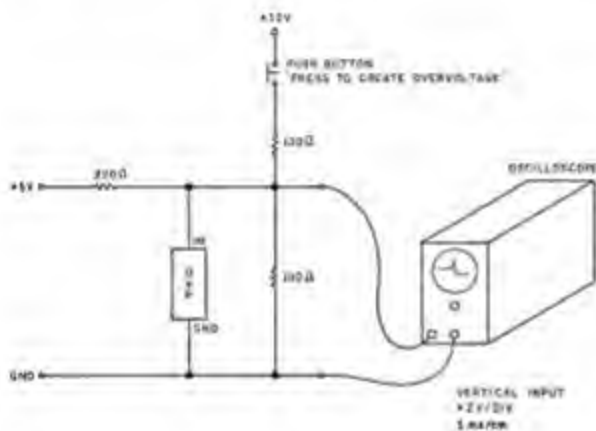


Figure 1.23 A test circuit to demonstrate the action of the overvoltage protector.

CHAPTER 2

CENTRAL PROCESSOR BASICS

There are many different microprocessors on the market and while instruction nomenclature is somewhat different for each one, the basic logical computing processes are similar in all devices. The rule to remember the next time a discussion turns to the capabilities of two computers is that "a computer is a computer." I don't wish to imply that they are all the same, but similarities abound and I would not like to spend a lifetime analyzing instruction sets and interfacing details before choosing one.

I once had lunch with the designer of one of the largest selling personal computer systems on the market. Thousands of computers had been sold, generating immense profits for the manufacturer. Our conversation eventually centered on the cost-effectiveness of his design. I had fanciful thoughts of a design team spending months reducing component count and analyzing instruction sets to determine minimum memory requirements. In actuality, my designer friend was given two months to come up with a manufacturable design. The investors' only question was the price and availability of the particular components he had chosen. Being an avid personal computer enthusiast, he simply built a computer around the microprocessor he already owned. The eventual advertising for his system touted the advanced architecture embodied in the central processor, but no machine-language programming facility was available to the user. It had only a high-level language BASIC interpreter and was, from an engineering point of view, simply a black-box computer. He could have used any microprocessor. So much for textbook engineering design.

Unfortunately, the hobbyist who is building a microcomputer from scratch, and who won't be making a black box, has to try to pick a device that is somewhere in the middle of the performance and capability spectrum. The general rule that all computers perform similar functions is true, but a printed-circuit board is a luxury. The hobbyist who has to do all the wiring by hand will surely be interested in efficient design. It's a fact that some of the more esoteric microprocessors require very expensive peripheral circuitry. Even devices that seem quite straightforward, with limited instruction sets, can require 50 or more ICs as interface elements. The ultimate configuration should be a trade-off between circuit complexity, ease of testing, and component price.

MICROPROCESSOR ARCHITECTURE

The internal architecture of the microprocessor determines the support devices required to make a microcomputer system. Perhaps the best place to start is to briefly discuss the major architectural differences.

Definition: A microcomputer is a logical machine that manipulates binary numbers (data) and processes this information by following an organized sequence of program steps referred to as instructions.

All microcomputers, like all computers, have the following features:

1. **Input** — Facilities must exist to allow the entrance of data or instructions.
2. **Memory** — The program sequence must be stored before and after execution, and resources must be available to store the result of any computations.
3. **Arithmetic logic unit** — Performs arithmetic operations on input or stored data.

4. **Control section** — Makes decisions regarding program flow and process control based on internal states of the results of arithmetic computations.
5. **Output** — The results are delivered to the user or stored in an appropriate medium.

The microprocessor is the single integrated circuit around which a microcomputer is constructed. *The microprocessor is a device; the microcomputer is a system.* In their least complex form, microprocessors include only the functions of items three and four and must rely on external devices attached to buses to perform the other tasks. Figure 2.1 is the basic block diagram of an 8-bit microcomputer and shows the interconnection of these buses and support elements. The computer in figure 2.1 uses six separate buses: memory address, memory data in and out, I/O address, and data input and output. The microprocessor contains a central processor that consists of the circuitry required to access the appropriate memory and I/O locations and interpret the resulting instructions that are also executed in this unit. The central processor also contains the ALU (Arithmetic and Logic Unit), which is a combination network that performs arithmetic and logical operations on the data. Additionally, the central processor includes a control section that governs the operations of the computer, and the various data registers used for manipulation and storage of data and instructions.

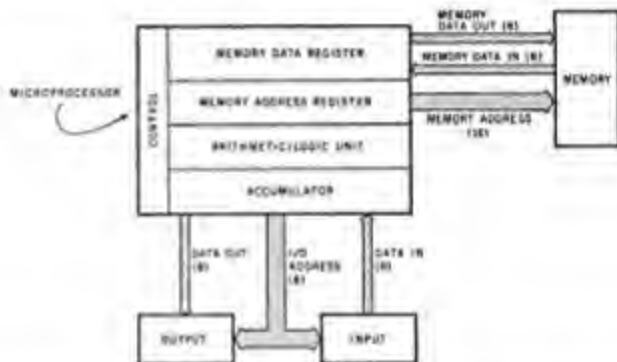


Figure 2.1 A basic block diagram of a microcomputer illustrating the data busing concept. Numbers in parentheses are the usual required quantity of physical wires to perform bus functions for an 8-bit microprocessor.

Actually few microprocessors support six separate buses. The number of pins that would be required on the IC is out of the question. Instead, to reduce pinouts, component manufacturers often combine the data input and output buses and make them "bi-directional." During an output instruction, data flows from the microprocessor to the output device and vice versa during an input instruction. To further cut the number of pins required on the central processor, the memory address bus can also serve as the address bus for input and output devices. During input/output instructions, the address present on the address lines references a particular input/output device(s). The resulting reduced configuration is shown in figure 2.2.

The concept of two buses is easy to understand and, from a hardware point of view, easy to utilize. The buses are time and function multiplexed. That is, during memory operations, the bits on the address bus refer to a memory location, and data on the data bus represent the content of memory. The direction of the data flow (to or from the central processor) is controlled within the microprocessor. Activities with input/output devices are performed in a similar fashion. During those instructions, input or output data and device addresses occupy the buses.

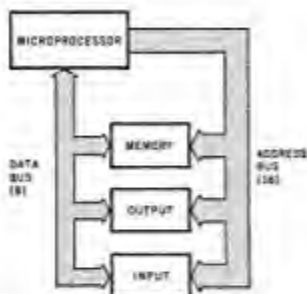


Figure 2.2 A block diagram of a microcomputer utilizing multiplexed bi-directional busing techniques to reduce pinout.

The number of bus wires can be further reduced by combining both data and address on the same lines and time multiplexing the data transfer along them. Figure 2.3 illustrates this final configuration. This method requires additional circuit elements to demultiplex and store pertinent data. The additional external components necessary to use this architectural feature defeat its purpose and make its use inadvisable for the hobbyist. There are other microprocessors that are simpler to use.

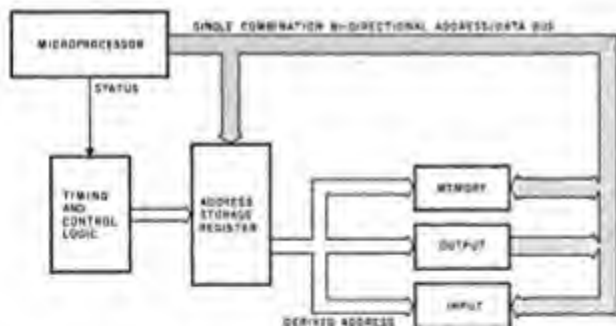


Figure 2.3 A block diagram of a microcomputer utilizing a single multiplexed bi-directional bus for both memory and input/output functions.

When building rather than buying a personal computer, the following criteria must be carefully considered:

1. **Circuit complexity** — Keep components to a reasonable minimum. The more components in a design, the more likelihood of wiring errors and faulty devices.
2. **Cost** — While cost is important, it should not be the primary consideration. Any microprocessor function could be simulated by using small scale integrated logic; however, indirect costs resulting from using 200 chips to replace 3 or 4 LSI (large scale integration) devices would negate the value of using cheaper parts initially. On the other hand, in the semiconductor industry, density means dollars. The more functions a device can provide, and the fewer components necessary to ac-

- compish these tasks, the higher the price. The level of integration incorporated in a homebrew computer should fit somewhere in the middle. The ZAP computer outlined in this book is a prime example of this philosophy. It uses a combination of cost-effective LSI (large scale integration) and inexpensive SSI (small scale integration) to produce a computer that the hobbyist can truly build, test, and use.
3. **Software compatibility and availability** — Building the hardware of a microcomputer is only half the job. It must be programmed to perform useful work. Initially, the builder will by necessity hand code and assemble his own programs. Eventually, however, the need may arise for the computer to do a task requiring a very large program which cannot be easily hand assembled. The user must rely upon an assembler program in a larger machine. The assembler program would, of course, have to be compatible with the instruction set of the microcomputer.

A further consideration is that personal computer enthusiasts are forever exchanging software. It is possible to convert programs to run on any central processor, but the effort would be the same as writing the entire program from scratch. This defeats the purpose of exchanging software. The personal computer owner should choose a microprocessor that is somewhat compatible with the computers already on the market. My statement that all computers are alike is theoretically true, but a book on how to build an esoteric one-of-a-kind computer is of little practical value.

Each criterion could be analyzed and answered individually, but we must give some credit to the manufacturers of personal computers for doing some of the thinking for us already. The fact that so many personal computers are in use has established *de facto* standardization of central processor choice. To be compatible with existing software and to have sufficient documentation available, the builder should consider choosing among those central processors in commercial use. The four most used microprocessors are

1. Intel 8080A
2. Motorola 6800
3. MOS Technology 6502
4. Zilog Z80

As a result of each device's wide following, documentation and software are readily available. The availability of 8080A compatible software is highest; cost is low, but its circuit complexity is also the greatest of the above. The 8080A, while described as a "single-chip computer," relies on various external drivers and support devices. Its minimum functional configuration consists of three chips as shown in figure 2.4. Its central processor bus structure is similar to figure 2.3, but when combined with the 8224 and 8228 support chips, it emulates the more desirable bus architecture outlined in figure 2.2.

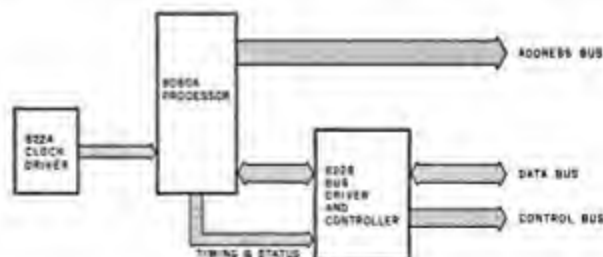


Figure 2.4 A minimum three-chip 8080A configuration illustrating the necessary support devices. The control bus contains the timing functions necessary to decode the contents of the data and address buses.

The best of both worlds is incorporated within the Z80. Not only does it execute the complete instruction set of the 8080A, but it also has additional instructions that serve to make it a very powerful processor. The Z80 bus structure is illustrated in figure 2.5. The Z80 is slightly more expensive than the other processors listed. However, its reduced external circuitry results in an effective cost comparison. Further, the ease of interfacing the Z80 makes it the natural choice when building a microcomputer from scratch.

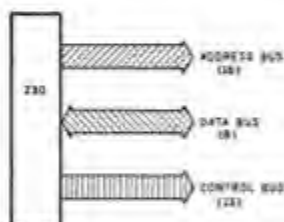


Figure 2.5 A block diagram of the Zilog Z80 bus structure.

CHAPTER 3

THE Z80

MICROPROCESSOR

Many books have been written on the software and hardware attributes of the Z80. Although I am not attempting to duplicate the efforts of other authors, any book dedicated to the construction of a microcomputer would be incomplete without a section describing the processor in some detail. By completely understanding the internal logic and external control functions of the central processor, you will be able to understand better the way I've designed the rest of the system hardware. You have many options when constructing a computer from scratch. The deeper your degree of understanding, the greater your confidence in the outcome, and it is more likely that you will add enhancements to your own design.

The ZAP computer allows considerable latitude in the selection of peripheral interfacing. The choice depends primarily upon the design philosophy of the system, which starts with the central processor.

CENTRAL PROCESSOR ARCHITECTURE

The Z80 is a register-oriented microprocessor. Eighteen 8-bit and four 16-bit registers within the central processor are accessible to the programmer and function as static programmable memory. These registers are divided into two sets, main and alternate, each of which contains six general purpose 8-bit registers that may be used either individually, or as three pairs of 16-bit registers. Also included are two sets of accumulators and flag registers. Figure 3.1 illustrates the internal architecture of the Z80 central processor. Figure 3.2 shows that within the Z80 there are accumulators and flag registers, along with general and special purpose registers.

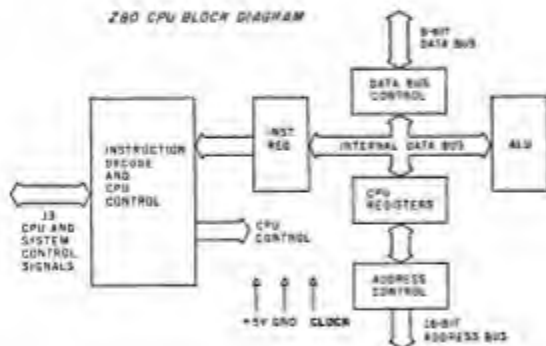


Figure 3.1 A block diagram of the internal architecture of the Z80 central processor.

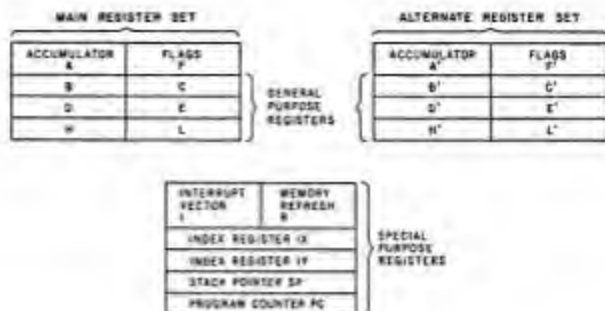


Figure 3.2 280 central processor register configuration.

The following is a description of the function and structure of the major components of the central processor.

1. Registers

A. Accumulators and Flag Registers

The central processor contains two independent accumulator and flag-register pairs, one in the main register set and the other in the alternate register set. The accumulator receives the results of all 8-bit arithmetic and logical operations, whereas the flag register indicates the occurrence of specific logical or arithmetic conditions in the processor such as parity, zero, sign, carry, and overflow. A single exchange instruction allows the programmer to select either accumulator or flag-register pair.

B. General Purpose Registers

There are two similar sets of general purpose registers. The main register set contains six 8-bit registers called B, C, D, E, H, and L; the alternate register set also contains six 8-bit registers referred to as B', C', D', E', H', and L'. For 16-bit operations, these registers can be grouped in 16-bit pairs (BC, DE, HL or BC', DE', HL'). A single exchange instruction allows the programmer to alternately choose between the register-pair sets.

C. Special Purpose Registers

1. PC (program counter)

The program counter contains a 16-bit address in memory from which the current instruction will be fetched. Following execution of the instruction, the PC counter is either incremented, if the program is to proceed to the next byte in memory, or the present PC contents are replaced with a new value, if a jump or call instruction is to be executed.

2. SP (stack pointer)

The Z80 allows several levels of subroutine nesting through use of a "stack" and a "stack pointer"; when certain instructions are executed, or when calls to subroutines are made, the PC counter and other pertinent data can be temporarily stored on a stack. A stack is a reserved area of several memory locations, the top of which is indicated by the contents of the stack pointer. That is to say, the stack pointer shows the address of the most recently made entry, because the memory locations are organized as a last-in, first-out file. By looking at particular entries in the stack,

the central processor returns to a main program regardless of the depth of nested subroutines. Theoretically, the stack could be 64 K bytes long; however, program space must not be overwritten by an expanding stack.

D. IX and IY Index Registers

These registers facilitate table data manipulation. They are two independent 16-bit registers that hold the base addresses used in indexed addressing modes, and point to locations in memory where pertinent data is to be stored or retrieved. Incorporated within the indexed instructions is a two's complement signed integer that specifies displacement from this base address.

E. Interrupt Page Address Register (I)

This is an 8-bit register that can be loaded with a page address of an interrupt service routine. During a mode 2 interrupt program, control will vector to this page address.

F. Memory Refresh Register (R)

To enable dynamic memories for the Z80, a 7-bit memory refresh register is automatically incremented after each instruction fetch.

II. Arithmetic and Logic Unit

Arithmetic manipulations and logical operations are handled eight bits at a time in the Z80 ALU (arithmetic and logic unit). The ALU communicates internally to the central processor registers and is not directly accessible by the programmer. The ALU performs the following operations:

LEFT or RIGHT SHIFT

INCREMENT

DECREMENT

ADD

SUBTRACT

AND

OR

EXCLUSIVE OR

COMPARE

SET BIT

RESET BIT

TEST BIT

III. Instruction Register and Central Processor Control

The instruction register holds the contents of the memory location addressed by the PC (program counter) and is loaded during the fetch cycle of each instruction. The central processor control unit executes the functions defined by the instruction in the instruction register and generates all control signals necessary to transmit the results to the proper registers.

IV. Central Processor Hardware

A. Figure 3.3 details the pinout of the Z80. It comes in an industry standard 40 pin dual in-line package. The following is a listing and explanation of the pin functions:

A_0-A_{15} Three-state output, active high. A_0-A_{15} constitute a 16-bit address bus. These signals provide the address for memory data exchanges (up to 64 K bytes) and for I/O device data exchanges. I/O addressing uses the eight lower address bits to allow the user to directly select up to 256 input or 256 output ports. A_0 is the least significant address bit. During refresh time, the lower seven bits contain a valid refresh address.

D_0-D_7 Three-state input/output, active high. D_0-D_7 constitute an 8-bit bi-directional data bus which is used for data exchanges with memory and I/O devices.

\overline{M} Output, active low, \overline{M} indicates that the current machine cycle is the operation-code fetch cycle of an in-

Cycle One) instruction execution. Note that during execution of 2-byte opcodes, \overline{MI} is generated as each opcode byte is fetched. These 2-byte opcodes always begin with CBH, DDH, EDH, or FDH. \overline{MI} also occurs with \overline{IORQ} to indicate an interrupt acknowledge cycle.

\overline{MREQ}
(Memory Request) Three-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory-read or memory-write operation.

\overline{IORQ}
(Input/Output Request) Three-state output, active low. The \overline{IORQ} signal indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. An \overline{IORQ} signal is also generated with an \overline{MI} signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt acknowledge operations may occur during \overline{MI} time while I/O operations are prohibited.

\overline{RD}
(Memory Read) Three-state output, active low. \overline{RD} indicates that the central processor wants to read from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the central processor data bus.

\overline{WR}
(Memory Write) Three-state output, active low. \overline{WR} indicates that the central processor data bus holds valid data to be stored in the addressed memory or I/O device.

\overline{RFSH}
(Refresh) Output, active low. \overline{RFSH} indicates that the lower seven bits of the address bus contain a refresh address for dynamic memories and the current \overline{MREQ} signal should be used to do a refresh read to all dynamic memories.

\overline{HALT}
(Halt State) Output, active low. \overline{HALT} indicates that the central processor has executed a HALT instruction and is awaiting either a nonmaskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the central processor executes NOPs (no operation) to maintain memory refresh activity.

\overline{WAIT}
(Wait) Input, active low. \overline{WAIT} indicates to the Z80 central processor that the addressed memory or I/O devices are not ready for a data transfer. The central processor continues to enter wait states as long as \overline{WAIT} is active; this signal allows memory or I/O devices to be synchronized to the central processor.

\overline{INT}
(Interrupt) Input, active low. The Interrupt request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop is enabled and if the \overline{BUSRQ} signal is not active. When the central processor accepts the interrupt, an acknowledge signal (\overline{IORQ} during \overline{MI} time) is sent out at the beginning of the next instruction cycle. The central processor can respond to an interrupt in the three different modes.

\overline{NMI}
(Non-Maskable) Input, negative edge triggered. The nonmaskable interrupt request line has a higher priority than \overline{INT} and is always recognized at the end of the current instruction,

Interrupt) regardless of the status of the interrupt-enable flip-flop. NMI forces the Z80 central processor to restart at location 0066₁₆. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that a BUSRQ will override an NMI.

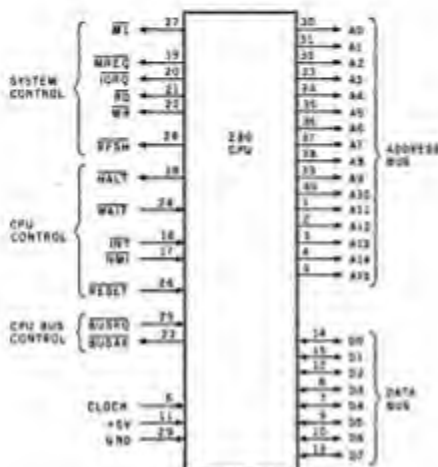


Figure 3.3 Pin configuration for the Z80 microprocessor.

The actual timing of these signals will be discussed in the hardware sections.

V. Z80 Instruction Types

The Z80 can execute 156 separate instructions including all 78 of the 8080A. They can be grouped as follows:

A. LOAD AND EXCHANGE

Load instructions move data between registers or between registers and memory. The source and destination of this data is specified within the instruction. Exchange instructions swap the contents of two registers.

B. ARITHMETIC AND LOGICAL

These instructions operate on data in the accumulator, a register, or a designated memory location. Results are placed in the accumulator and flags are set accordingly. Arithmetic operations include 16-bit addition and subtraction between register pairs.

C. BLOCK TRANSFER AND SEARCH

The Z80 uses a single instruction to transfer any size block of memory to any other group of contiguous memory locations. The block search uses a single command to examine a block of memory for a particular 8-bit character.

D. ROTATE AND SHIFT

Data can be rotated and shifted in the accumulator, a central processor register, or memory. These instructions also have binary-coded

decimal (BCD) handling facilities.

E. BIT MANIPULATION

Bit manipulation includes set, reset, and test functions. Individual bits may be modified or tested in the accumulator, a central processor, or memory. The results of the test operations are indicated in the flag register.

F. JUMP, CALL AND RETURN

A jump is a branch to a program location specified by the contents of the program counter. The program counter contents can come from three addressing modes: immediate, extended, or register indirect. A call is a special form of jump where the address following the call instruction is pushed onto the stack before the jump is made. A return is the reverse of the call. This category includes special restart instructions.

G. INPUT AND OUTPUT

These instructions transfer data between register and memory to external I/O devices. There are 256 input and 256 output ports available. Special instructions provide for moving blocks of 256 bytes to or from I/O ports and memory.

H. CPU CONTROL

These instructions include halting the CPU or causing a NOP (no operation) to be executed. The ability to enable or disable interrupt inputs is a further control capability.

VI. Instruction and Data Formats

Memory for the Z80 is organized into 8-bit quantities called bytes (see figure 3.4). Each program byte is stored in a unique memory position and is referenced by a 16-bit binary address.

Total direct addressing capability is 65,536 bytes (64 K) of memory, which may be any combination of ROM (read-only memory), EPROM (erasable-programmable read-only memory), or programmable memory. Data is stored in the format of figure 3.5.

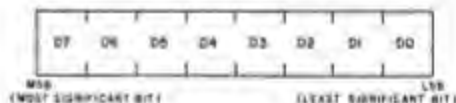


Figure 3.4 Organization of a data byte in the Z80.

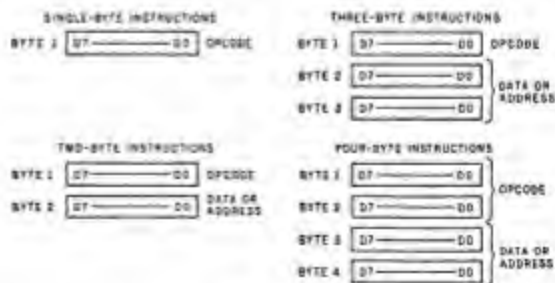


Figure 3.5 Machine-language instruction formats for the Z80.

VII. Z80 Status Flags

The flag register (F and F') supplies information to the user regarding the status of the central processor at any given time. There are four testable and two nontestable flag bits in each register. Figure 3.6 shows the position and identity of these flag bits.

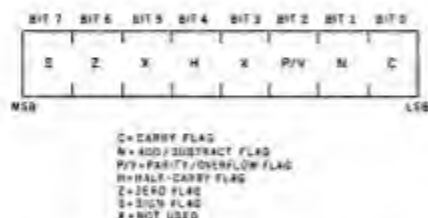


Figure 3.6 Position and identity of status flag bits in the flag register.

Instructions set (flag bit = 1) or reset (flag bit = 0) flags in a manner relevant to the particular operation being executed.

VIII. The Z80 Instruction Set

The following symbols and abbreviations are used in the subsequent description of the Z80 instructions:

Symbol	Meaning
accumulator	Register A
address	A 16-bit address quantity
high-order address	The most significant 8 bits of the 16-bit address
low-order address	The least significant 8 bits of the 16-bit address
data	An 8- or 16-bit quantity
high-order data	The most significant 8 bits of the 16-bit data
low-order data	The least significant 8 bits of the 16-bit data
port	An 8-bit address of an I/O device
r, r'	One of the registers A, B, C, D, E, H, or L
n	A 1-byte expression in the range of 0 thru 255
nn	A 2-byte expression in the range of 0 thru 65,535
d	A 1-byte expression in the range of -128 to 127
b	An expression in the range of 0 thru 7
e	A 1-byte expression in a range of -126 to 129
cc	The state of the flags for conditional JR and JP instructions:

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

XXH	Denotes hexadecimal address value
qq	Any one of the register pairs BC, DE, HL, or AF
ss	Any one of the register pairs BC, DE, HL, or SP

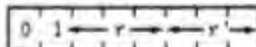
pp	Any one of the register pairs BC, DE, IX, or SP
rr	Any one of the register pairs BC, DE, IX, or SP
s	Any of r, n, (HL), (IX+d), or (IY+d)
dd	Any one of the register pairs BC, DE, HL, or SP
m	Any of r, (HL), (IX+d), or (IY+d)
(HL)	Specifies the contents of memory at the location addressed by the contents of the register pair HL
(nn)	Specifies the contents of memory at the location addressed by the 2-byte expression in nn
PC	Program counter
SP	Stack pointer
t	An expression in the range of 0 thru 7.
C, N, P/V, H, Z, S	Condition flags:
	C Carry
	N Add/Subtract
	P/V Parity/Overflow
	H Half-Carry
	Z Zero
	S Sign
←	"is transferred to"
^	Logical AND
⊕	Exclusive OR
∨	Inclusive OR
+	Addition
-	Subtraction
↔	"is exchanged with"

EIGHT-BIT LOAD GROUP

LD r, r'

$r \leftarrow r'$

The contents of any register r' are loaded into any other register r.

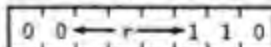


Cycles: 1
States: 4
Flags: none

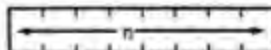
LD r, n

$r \leftarrow n$

The 8-bit integer n is loaded into any register r.



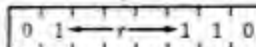
Cycles: 2
States: 7
Flags: none



LD r, (HL)

$r \leftarrow (HL)$

The 8-bit contents of memory location (HL) are loaded into register r.



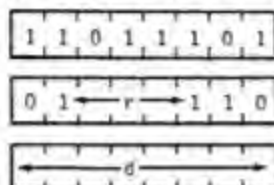
Cycles: 2
 States: 7
 Flags: none

LD r, (IX+d)

$r \leftarrow (IX+d)$

The operand (IX+d) (the contents of the Index Register IX summed with a displacement integer d) is loaded into register r.

Cycles: 5
 States: 19
 Flags: none

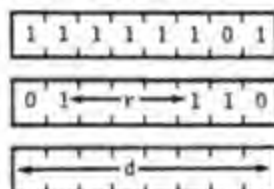


LD r, (IY+d)

$r \leftarrow (IY+d)$

The operand (IY+d) (the contents of the Index Register IY summed with a displacement integer d) is loaded into register r.

Cycles: 5
 States: 19
 Flags: none

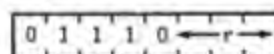


LD (HL), r

$(HL) \leftarrow r$

The contents of register r are loaded into the memory location specified by the contents of the HL register pair.

Cycles: 2
 States: 7
 Flags: none

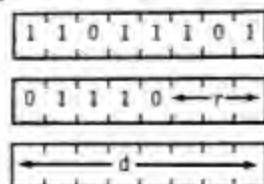


LD (IX+d), r

$(IX+d) \leftarrow r$

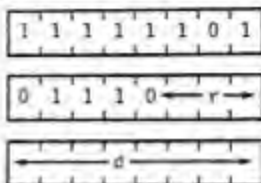
The contents of register r are loaded into the memory address specified by the contents of Index Register IX summed with d, which is a two's complement displacement integer.

Cycles: 5
 States: 19
 Flags: none



LD (IY+d), r**(IY+d) ← r**

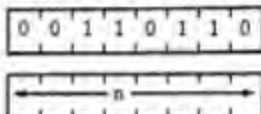
The contents of register *r* are loaded into the memory address specified by the sum of the contents of the Index Register IY and *d*, a two's complement displacement integer.



Cycles: 5
States: 19
Flags: none

LD (HL), n**(HL) ← n**

Integer *n* is loaded into the memory address specified by the contents of the HL register pair.



Cycles: 3
States: 10
Flags: none

LD (IX+d), n**(IX+d) ← n**

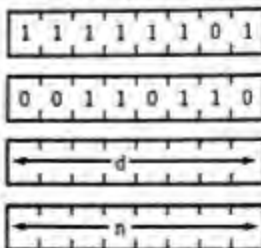
The *n* operand is loaded into the memory address specified by the sum of the contents of the Index Register IX and the two's complement displacement operand *d*.



Cycles: 5
States: 19
Flags: none

LD (IY+d), n**(IY+d) ← n**

Integer *n* is loaded into the memory location specified by the contents of the Index Register IY summed with a displacement integer *d*.

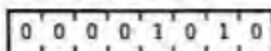


Cycles: 5
States: 19
Flags: none

LD A, (BC)

A ← (BC)

The contents of the memory location specified by the contents of the BC register pair are loaded into the Accumulator.

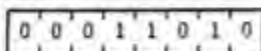


Cycles: 2
States: 7
Flags: none

LD A, (DE)

A ← (DE)

The contents of the memory location specified by the register pair DE are loaded into the Accumulator.

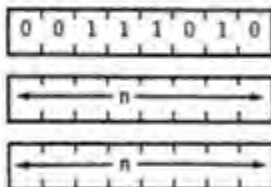


Cycles: 2
States: 7
Flags: none

LD A, (nn)

A ← (nn)

The contents of the memory location specified by the operands nn are loaded into the Accumulator. The first n operand is the low-order byte of a 2-byte memory address.

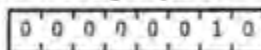


Cycles: 4
States: 13
Flags: none

LD (BC), A

(BC) ← A

The contents of the Accumulator are loaded into the memory location specified by the contents of the register pair BC.

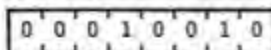


Cycles: 2
States: 7
Flags: none

LD (DE), A

(DE) ← A

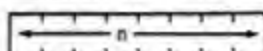
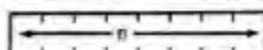
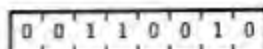
The contents of the Accumulator are loaded into the memory location specified by the DE register pair.



Cycles: 2
States: 7
Flags: none

LD (nn), A**(nn) ← A**

The contents of the Accumulator are loaded into the memory address specified by the operands nn. The first n operand is the low-order byte of operand nn.



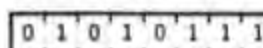
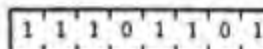
Cycles: 4

States: 13

Flags: none

LD A, I**A ← I**

The contents of the Interrupt Vector Register I are loaded into the Accumulator.



Cycles: 2

States: 9

Flags: S, Z, H, N, P/V

S: set if I < 0; reset otherwise

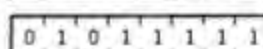
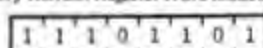
Z: set if I = 0; reset otherwise

H, N: reset

P/V: contains contents of IFF2

LD A, R**A ← R**

The contents of Memory Refresh Register R are loaded into the Accumulator.



Cycles: 2

States: 9

Flags: S, Z, H, N, P/V

S: set if R < 0; reset otherwise

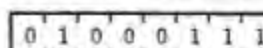
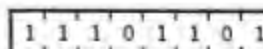
Z: set if R = 0; reset otherwise

H, N: reset

P/V: contains contents of IFF2

LD I, A**I ← A**

The contents of the Accumulator are loaded into the Interrupt Control Vector Register I.



Cycles: 2

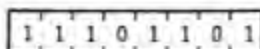
States: 9

Flags: none

LD R, A

R ← A

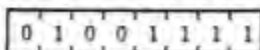
The contents of the Accumulator are loaded into the Memory Refresh Register R.



Cycles: 2

States: 9

Flags: none



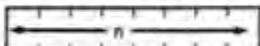
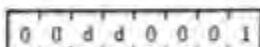
SIXTEEN-BIT LOAD INSTRUCTIONS

LD dd, nn

dd ← nn

The 2-byte integer nn is loaded into the dd register pair, where dd defines the BC, DE, HL, or SP register pairs, assembled as follows in the object code:

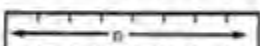
Pair	dd
BC	00
DE	01
HL	10
SP	11



Cycles: 3

States: 10

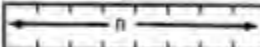
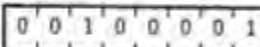
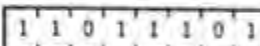
Flags: none



LD IX, nn

IX ← nn

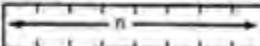
Integer nn is loaded into the Index Register IX.



Cycles: 4

States: 14

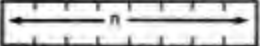
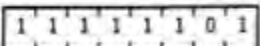
Flags: none



LD IY, nn

IY ← nn

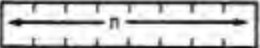
Integer nn is loaded into the Index Register IY.



Cycles: 4

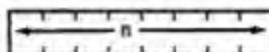
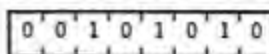
States: 14

Flags: none



LD HL, (nn) $H \leftarrow (nn+1), L \leftarrow (nn)$

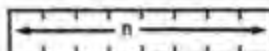
The contents of memory address nn are loaded into register L , and the contents of the next highest memory location $(nn+1)$ are loaded into register H .



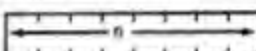
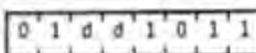
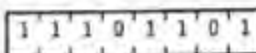
Cycles: 5

States: 16

Flags: none

**LD dd, (nn)** $dd_H \leftarrow (nn+1), dd_L \leftarrow (nn)$

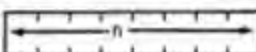
The contents of address nn are loaded into the low-order portion of register pair dd , and the contents of the next highest memory address $(nn+1)$ are loaded into the high portion of dd .



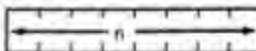
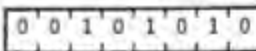
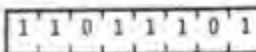
Cycles: 6

States: 20

Flags: none

**LD IX, (nn)** $IX_H \leftarrow (nn+1), IX_L \leftarrow (nn)$

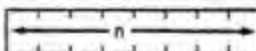
The contents of the address nn are loaded into the low-order portion of Index Register IX , and the contents of the next highest memory address $(nn+1)$ are loaded into the high-order portion of IX .



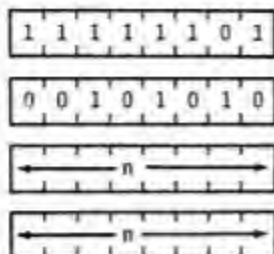
Cycles: 6

States: 20

Flags: none

**LD IY, (nn)** $IY_H \leftarrow (nn+1), IY_L \leftarrow (nn)$

The contents of address nn are loaded into the low-order portion of Index Register IY , and the contents of the next highest memory address $(nn+1)$ are loaded into the high-order portion of IY .



Cycles: 6
States: 20
Flags: none

LD (nn), HL

$(nn+1) \leftarrow H, (nn) \leftarrow L$

The contents of register L are loaded into memory address nn , and the contents of register H are loaded into the next highest address location $nn+1$.

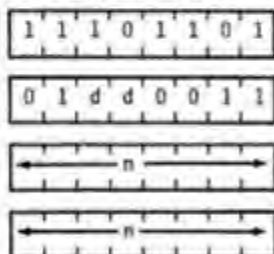


Cycles: 5
States: 16
Flags: none

LD (nn), dd

$(nn+1) \leftarrow dd_H, (nn) \leftarrow dd_L$

The low-order byte of register pair dd is loaded into memory address nn ; the upper byte is loaded into memory address $nn+1$.

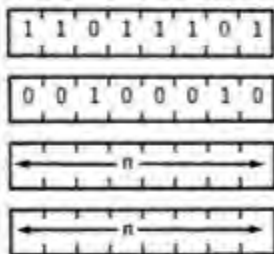


Cycles: 6
States: 20
Flags: none

LD (nn), IX

$(nn+1) \leftarrow IX_H, (nn) \leftarrow IX_L$

The low-order byte in Index Register IX is loaded into memory address nn ; the upper-order byte is loaded into the next highest address $nn+1$.



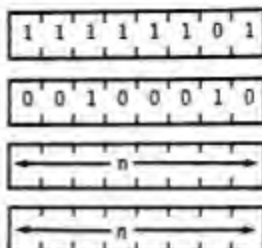
Cycles: 6
States: 20
Flags: none

LD (nn), IY

$(nn+1) \leftarrow IY_H, (nn) \leftarrow IY_L$

The low-order byte in Index Register IY is loaded into memory address nn ; the upper-order byte is loaded into memory location $nn+1$.

Cycles: 6
States: 20
Flags: none

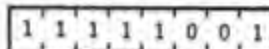


LD SP, HL

SP ← HL

The contents of the register pair HL are loaded into the SP (stack pointer).

Cycles: 1
States: 6
Flags: none

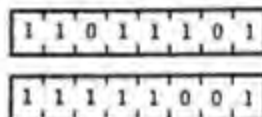


LD SP, IX

SP ← IX

The 2-byte contents of Index Register IX are loaded into the SP (stack pointer).

Cycles: 2
States: 10
Flags: none



LD SP, IY

SP ← IY

The 2-byte contents of Index Register IY are loaded into the SP (stack pointer).

Cycles: 2
States: 10
Flags: none

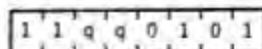


PUSH qq

(SP-2) ← qq_h, (SP-1) ← qq_l

The contents of the register pair qq are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of register pair qq into the memory address now specified by the SP; then decrements the SP again and loads the low order byte of qq into the memory location corresponding to this new address in the SP.

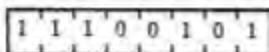
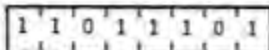
Cycles: 3
States: 11
Flags: none



PUSH IX

$$(SP-2) \leftarrow IX_L, (SP-1) \leftarrow IX_H$$

The contents of the Index Register IX are pushed into the Stack. This instruction first decrements the SP and loads the high-order byte of IX into the memory address now specified by the SP; it then decrements the SP again and loads the low-order byte into the memory location corresponding to this new address in the SP.

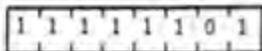


Cycles: 3
States: 15
Flags: none

PUSH IY

$$(SP-2) \leftarrow IY_L, (SP-1) \leftarrow IY_H$$

The contents of the Index Register IY are pushed into the Stack. This instruction first decrements the SP and loads the high-order byte of IY into the memory address now specified by the SP; it then decrements the SP again and loads the low-order byte into the memory location corresponding to this new address in the SP.

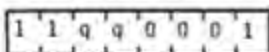


Cycles: 4
States: 15
Flags: none

POP qq

$$qq_H \leftarrow (SP+1), qq_L \leftarrow (SP)$$

The top 2 bytes of the Stack are popped into register pair qq. This instruction first loads into the low-order portion of qq the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high-order portion of qq, and the SP is now incremented again.

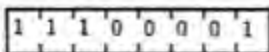
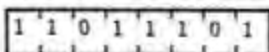


Cycles: 3
States: 10
Flags: none

POP IX

$$IX_H \leftarrow (SP+1), IX_L \leftarrow (SP)$$

The top 2 bytes of the Stack are popped into Index Register IX. This instruction first loads into the low-order portion of IX the byte at the memory location corresponding to the contents of SP; the SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high-order portion of IX. The SP is now incremented again.



Cycles: 4
States: 14
Flags: none

POP IY

$IY_H \leftarrow (SP+1)$, $IY_L \leftarrow (SP)$

The top 2 bytes of the Stack are popped into Index Register IY. This instruction first loads into the low-order portion of IY the byte at the memory location corresponding to the contents of SP; then the SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high-order portion of IY. The SP is now incremented again.

1	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

1	1	1	0	0	0	0	1
---	---	---	---	---	---	---	---

Cycles: 4
States: 14
Flags: none

EXCHANGE, BLOCK TRANSFER AND SEARCH GROUP

EX DE, HL

$DE \leftrightarrow HL$

The 2-byte contents of register pairs DE and HL are exchanged.

1	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Cycles: 1
States: 4
Flags: none

EX AF, AF'

$AF \leftrightarrow AF'$

The 2-byte contents of the register pairs AF and AF' are exchanged.

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

Cycles: 1
States: 4
Flags: none

EXX

$(BC) \leftrightarrow (BC')$, $(DE) \leftrightarrow (DE')$, $(HL) \leftrightarrow (HL')$

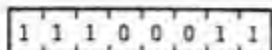
Each 2-byte value in register pairs BC, DE, and HL is exchanged with the 2-byte value in BC', DE', and HL' respectively.

1	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---

Cycles: 1
States: 4
Flags: none

EX (SP), HL $H \leftrightarrow (SP+1), L \leftrightarrow (SP)$

The low-order byte contained in register pair HL is exchanged with the contents of the memory address specified by the contents of register pair SP, and the high-order byte of HL is exchanged with the next highest memory address (SP+1).



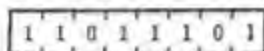
Cycles: 5

States: 19

Flags: none

EX (SP), IX $IX_H \leftrightarrow (SP+1), IX_L \leftrightarrow (SP)$

The low-order byte in the Index Register IX is exchanged with the contents of the memory address specified by the contents of register pair SP, and the high-order byte of IX is exchanged with the next highest address (SP+1).



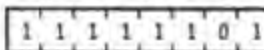
Cycles: 6

States: 23

Flags: none

EX (SP), IY $IY_H \leftrightarrow (SP+1), IY_L \leftrightarrow (SP)$

The low-order byte in Index Register IY is exchanged with the contents of the memory address specified by the contents of register pair SP, and the high-order byte of IY is exchanged with the next highest memory address.



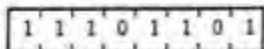
Cycles: 6

States: 23

Flags: none

LDI $(DE) \leftarrow (HL), DE \leftarrow DE+1, HL \leftarrow HL+1, BC \leftarrow BC-1$

A byte of data is transferred from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both register pairs are incremented and the BC (byte counter) register pair is decremented.



Cycles: 4

States: 16

Flags: H, N, P/V

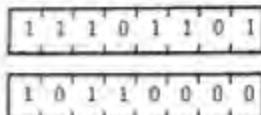
H, N: reset

P/V: set if $BC-1 \neq 0$; reset otherwise

LDIR

(DE) ← (HL), DE ← DE+1, HL ← HL+1, BC ← BC-1

This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Then, both register pairs are incremented and the BC (byte counter) register pair is decremented. If decrementing causes the BC to go to 0, the instruction is terminated. If BC is not 0, the program counter is decremented by 2 and the instruction is repeated. Note: if BC is set to 0 prior to instruction execution, the instruction will loop through 64 K bytes. Also, interrupts will be recognized after each data transfer.



For BC ≠ 0:

Cycles: 5

States: 21

For BC = 0:

Cycles: 4

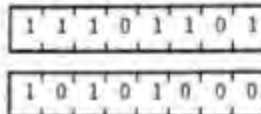
States: 16

Flags: H, N, P/V: reset

LDD

(DE) ← (HL), DE ← DE-1, HL ← HL-1, BC ← BC-1

This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both register pairs including the BC (byte counter) register pair are decremented.



Cycles: 4

States: 16

Flags: H, N, P/V

H, N: reset

P/V: set if BC-1 ≠ 0; reset otherwise

LDDR

(DE) ← (HL), DE ← DE-1, HL ← HL-1, BC ← BC-1

This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the DE register pair to the memory location addressed by the contents of the HL register pair. Then both registers, as well as the BC (byte counter), are decremented. If decrementing causes the BC to go to 0, the instruction is terminated. If BC is not 0, the program counter is decremented by 2 and the instruction is repeated. Note: if BC is set to 0 prior to instruction execution, the instruction will loop through 64 K bytes. Also, interrupts will be recognized after each data transfer.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

1	0	1	1	1	0	0	0
---	---	---	---	---	---	---	---

For $BC \neq 0$:

Cycles: 5

States: 21

For $BC = 0$:

Cycles: 4

States: 16

Flags: H, N, P/V: reset

CPI

$A \leftarrow (HL)$, $HL \leftarrow HL + 1$, $BC \leftarrow BC - 1$

The contents of the memory location addressed by the HL register pair are compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. Then HL is incremented and the byte counter (register pair BC) is decremented.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

1	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---

Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if $A = (HL)$; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

P/V: set if $BC - 1 \neq 0$; reset otherwise

CPIR

$A \leftarrow (HL)$, $HL \leftarrow HL + 1$, $BC \leftarrow BC - 1$

The contents of the memory location addressed by the HL register are compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL is incremented and the BC is decremented. If decrementing causes the BC to go to 0 or if $A = (HL)$, the instruction is terminated. If BC is not 0 and if $A \neq (HL)$, the program counter is decremented by two, and the instruction is repeated. Note: if BC is set to 0 before instruction execution, the instruction will loop through 64 K bytes, if no match is found. Also, interrupts will be recognized after each data comparison.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

1	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---

For $BC \neq 0$ and $A \neq (HL)$:

Cycles: 5

States: 21

For $BC = 0$ or $A = (HL)$:

Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if $A = (HL)$; reset otherwise

H: set if no borrow from bit 4; reset otherwise

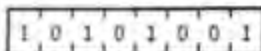
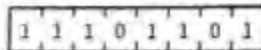
N: set

P/V: set if $BC - 1 \neq 0$; reset otherwise

CPD

$A \leftarrow (HL)$, $HL \leftarrow HL - 1$, $BC \leftarrow BC - 1$

The contents of the memory location addressed by the HL register pair are compared with the contents of the Accumulator. In case of a true compare a condition bit is set. The HL and the BC are decremented.



Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if $A = (HL)$; reset otherwise

H: set if no borrow from bit 4; reset otherwise

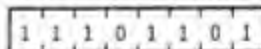
N: set

P/V: set if $BC - 1 \neq 0$; reset otherwise

CPDR

$A \leftarrow (HL)$, $HL \leftarrow HL - 1$, $BC \leftarrow BC - 1$

The contents of the memory location addressed by the HL register pair are compared with the contents of the Accumulator. In case of a true compare a condition bit is set. The HL and BC register pairs are decremented. If decrementing causes the BC to go to 0 or if $A = (HL)$, the instruction is terminated. If BC is not 0 and $A \neq (HL)$, the program counter is decremented by 2 and the instruction is repeated. Note: if BC is set to 0 prior to instruction execution, the instruction will loop through 64 K bytes if no match is found. Also, interrupts will be recognized after each data comparison.



For $BC \neq 0$ and $A \neq (HL)$:

Cycles: 5

States: 21

For $BC = 0$ or $A = (HL)$:

Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if $A = (HL)$; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

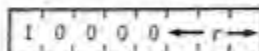
P/V: set if $BC - 1 \neq 0$; reset otherwise

EIGHT-BIT ARITHMETIC AND LOGICAL GROUP

ADD A, r

$$A \leftarrow A + r$$

The contents of register *r* are added to the contents of the Accumulator, and the result is stored in the Accumulator.



Cycles: 1

States: 4

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: reset

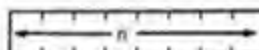
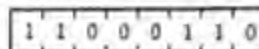
C: set if carry from bit 7; reset otherwise

P/V: set if overflow; reset otherwise

ADD A, n

$$A \leftarrow A + n$$

The integer *n* is added to the contents of the Accumulator, and the results are stored in the Accumulator.



Cycles: 2

States: 7

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: reset

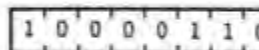
C: set if carry from bit 7; reset otherwise

P/V: set if overflow; reset otherwise

ADD A, (HL)

$$A \leftarrow A + (HL)$$

The byte at the memory address specified by the contents of the HL register pair is added to the contents of the Accumulator, and the result is stored in the Accumulator.



Cycles: 2

States: 7

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

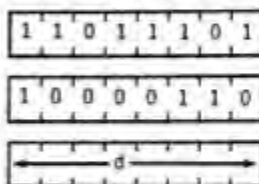
N: reset

C: set if carry from bit 7; reset otherwise

P/V: set if overflow; reset otherwise

ADD A, (IX+d)**A ← A + (IX+d)**

The contents of the Index Register IX are added to a displacement *d* to point to an address in memory. The contents of this address are then added to the contents of the Accumulator, and the result is stored in the Accumulator.



Cycles: 5

States: 19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: reset

C: set if carry from bit 7; reset otherwise

P/V: set if overflow; reset otherwise

ADD A, (IY+d)**A ← A + (IY+d)**

The contents of the Index Register IY are added to a displacement *d* to point to an address in memory. The contents of this address are then added to the contents of the Accumulator, and the result is stored in the Accumulator.



Cycles: 5

States: 19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: set

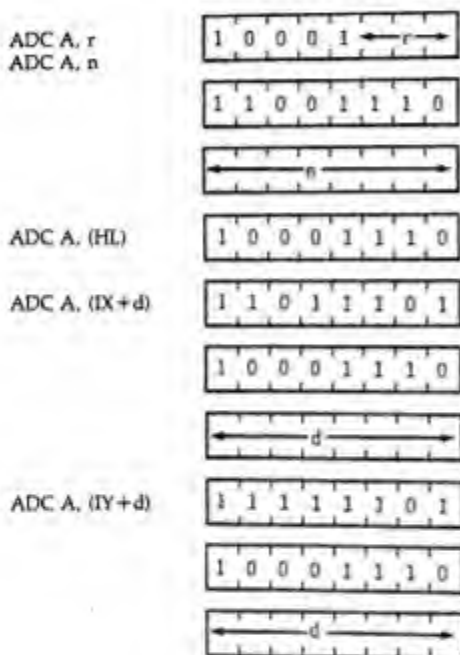
C: set if carry from bit 7; reset otherwise

P/V: set if overflow; reset otherwise

ADC A, s

$$A \leftarrow A + s + CY$$

The *s* operand is any of *r*, *n*, (HL), (IX+d), or (IY+d) as defined for the analogous ADD instruction. These various possible opcode operand combinations are assembled in the object code as follows:



The *s* operand, along with the Carry Flag ("C" in the F register) is added to the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	Cycles	States
ADC A, <i>r</i>	1	4
ADC A, <i>n</i>	2	7
ADC A, (HL)	2	7
ADC A, (IX+d)	5	19
ADC A, (IY+d)	5	19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: reset

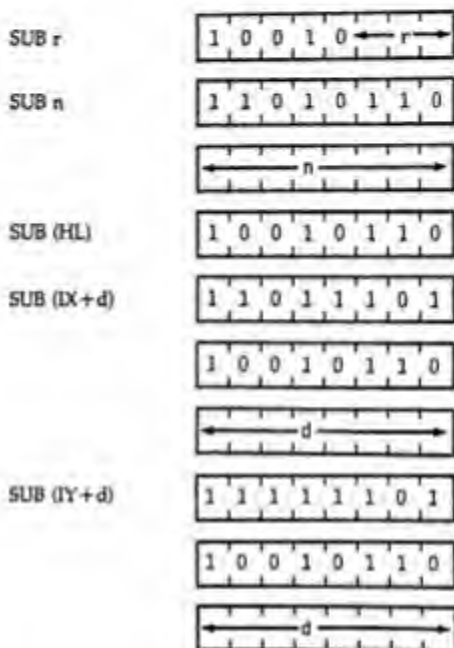
C: set if carry from bit 7; reset otherwise

P/V: set if overflow; reset otherwise

SUB s

$$A \leftarrow A - s$$

The *s* operand is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.



<u>Instruction</u>	<u>Cycles</u>	<u>States</u>
SUB r	1	4
SUB n	2	7
SUB (HL)	2	7
SUB (IX+d)	5	19
SUB (IY+d)	5	19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

C: set if no borrow; reset otherwise

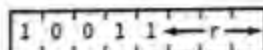
P/V: set if overflow; reset otherwise

SBC A, s

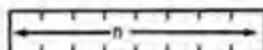
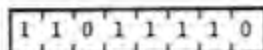
$A \leftarrow A - s - CY$

The s operand, along with the Carry Flag ("C" in the F register) is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

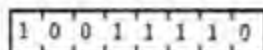
SBC A, r



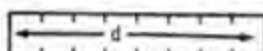
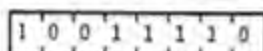
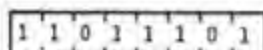
SBC A, n



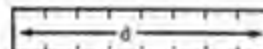
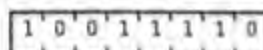
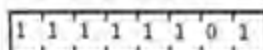
SBC A, (HL)



SBC A, (IX+d)



SBC A, (IY+d)



Instruction	Cycles	States
SBC A, r	1	4
SBC A, n	2	7
SBC A, (HL)	2	7
SBC A, (IX+d)	5	19
SBC A, (IY+d)	5	19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

C: set if no borrow; reset otherwise

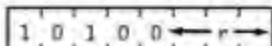
P/V: set if overflow; reset otherwise

AND s

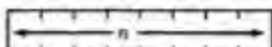
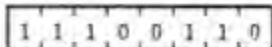
$A \leftarrow A \wedge s$

A logical AND operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

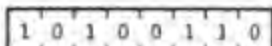
AND r



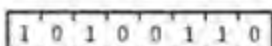
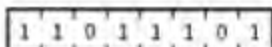
AND n



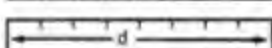
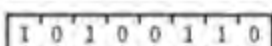
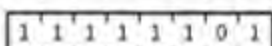
AND (HL)



AND (IX+d)



AND (IY+d)



Instruction

Cycles

States

AND r

1

4

AND n

2

7

AND (HL)

2

7

AND (IX+d)

5

19

AND (IY+d)

5

19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set

N: reset

C: reset

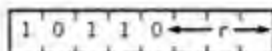
P/V: set if parity even; reset otherwise

OR s

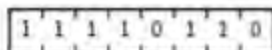
$A \leftarrow A \vee s$

A logical OR operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

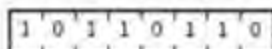
OR r



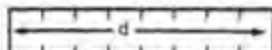
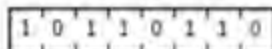
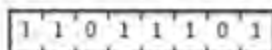
OR n



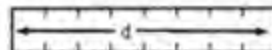
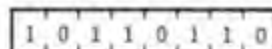
OR (HL)



OR (IX+d)



OR (IY+d)



Instruction	Cycles	States
OR r	1	4
OR n	2	7
OR (HL)	2	7
OR (IX+d)	5	19
OR (IY+d)	5	19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set

N: reset

C: reset

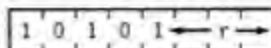
P/V: set if parity even; reset otherwise

XOR s

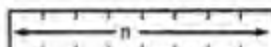
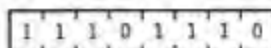
$A \leftarrow A \oplus s$

A logical exclusive-OR operation, bit by bit, is performed between the byte specified by the *s* operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

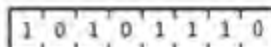
XOR *r*



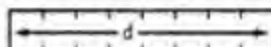
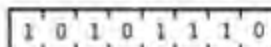
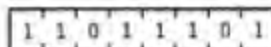
XOR *n*



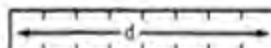
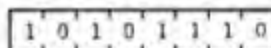
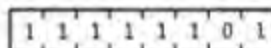
XOR (HL)



XOR (IX+d)



XOR (IY+d)



Instruction	Cycles	States
XOR <i>r</i>	1	4
XOR <i>n</i>	2	7
XOR (HL)	2	7
XOR (IX+d)	5	19
XOR (IY+d)	5	19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set

N: reset

C: reset

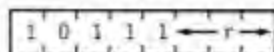
P/V: set if parity even; reset otherwise

CP s

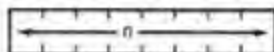
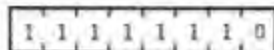
A-s

The contents of the s operand are compared with the contents of the Accumulator. If there is a true compare, a flag is set.

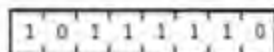
CP r



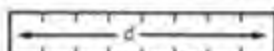
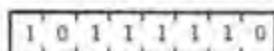
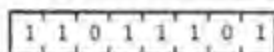
CP n



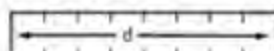
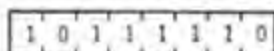
CP (HL)



CP (IX+d)



CP (IY+d)



Instruction	Cycles	States
CP r	1	4
CP n	2	7
CP (HL)	2	7
CP (IX+d)	5	19
CP (IY+d)	5	19

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

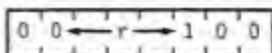
C: set if no borrow; reset otherwise

P/V: set if overflow; reset otherwise

INC r

$$r = r + 1$$

Register r is incremented.



Cycles: 1

States: 4

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

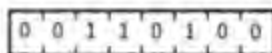
N: reset

P/V: set if r was 7FH before operation; reset otherwise

INC (HL)

$$(HL) \leftarrow (HL) + 1$$

The byte contained in the address specified by the contents of the HL register pair is incremented.



Cycles: 3

States: 11

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

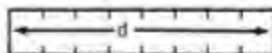
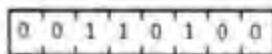
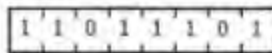
N: reset

P/V: set if (HL) was 7FH before operation; reset otherwise

INC (IX+d)

$$(IX+d) \leftarrow (IX+d) + 1$$

The contents of the Index Register IX are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.



Cycles: 6

States: 23

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: reset

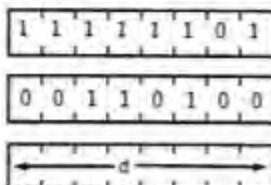
P/V: set if (IX+d) was 7FH before operation; reset otherwise

INC (IY+d)

$$(IY+d) \leftarrow (IY+d) + 1$$

The contents of the Index Register IY are added to a two's complement

displacement integer d to point to an address in memory. The contents of this address are then incremented.



Cycles: 6

States: 23

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if carry from bit 3; reset otherwise

N: reset

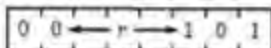
P/V: set if (IX+d) was 7FH before operation; reset otherwise

DEC m

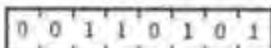
$m \leftarrow m-1$

The byte specified by the m operand is decremented.

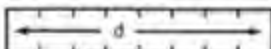
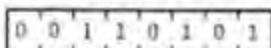
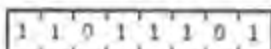
DEC r



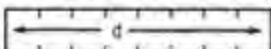
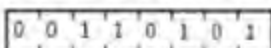
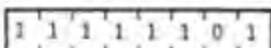
DEC (HL)



DEC (IX+d)



DEC (IY+d)



Instruction

Cycles

States

DEC r

1

4

DEC (HL)

3

11

DEC (IX+d)

6

23

DEC (IY+d)

6

23

Flags: S, Z, H, N, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

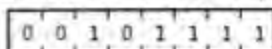
P/V: set if m was 80H before operation; reset otherwise

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

CPL

$$A \leftarrow \overline{A}$$

Contents of the Accumulator are inverted (1's complement).



Cycles: 1

States: 4

Flags: H, N

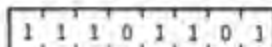
H: set

N: set

NEG

$$A \leftarrow 0 - A$$

The contents of the Accumulator are negated (two's complement). This is the same as subtracting the contents of the Accumulator from 0.



Cycles: 2

States: 8

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

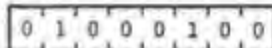
Z: set if result is 0; reset otherwise

H: set if no borrow from bit 4; reset otherwise

N: set

C: set if Accumulator was not 00H before operation; reset otherwise

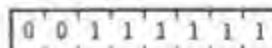
P/V: set if Accumulator was 80H before operation; reset otherwise



CCF

$$CY \leftarrow \overline{CY}$$

The C flag in the F register is inverted.



Cycles: 1

States: 4

Flags: H, N, C

H: previous carry will be copied

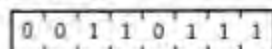
N: reset

C: set if CY was 0 before operation; reset otherwise

SCF

$$CY \leftarrow 1$$

The C flag in the F register is set.



Cycles: 1

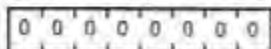
States: 4

Flags: H, N, C

H: reset
N: reset
C: set

NOP

The central processor performs no operation during this machine cycle.



Cycles: 1
States: 4
Flags: none

DAA

This instruction conditionally adjusts the Accumulator for BCD addition and subtraction operations. For addition (ADD, ADC, INC) or subtraction (SUB, SBC, DEC, NEG), the following table indicates the operation performed:

OPERATION	C BEFORE DAA	HEX VALUE IN UPPER DIGIT (bit 7-4)	H BEFORE DAA	HEX VALUE IN LOWER DIGIT (bit 3-0)	NUMBER ADDED TO BYTE	C AFTER DAA
ADD ADC INC	0	0-9	0	0-9	00	0
	0	0-9	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB SBC DEC NEG	0 0 1 1	0-9 0-9 7-F 6-F	0 1 0 1	0-9 6-F 0-9 6-F	00 FA A0 9A	0 0 1 1

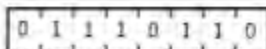
M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Cycles: 1
States: 4
Flags: S, Z, H, C, P/V

S: set if most significant bit of Accumulator is 1 after operation; reset otherwise
Z: set if Accumulator is 0 after operation; reset otherwise
H: see instruction
C: see instruction
P/V: set if Accumulator is even parity after operation; reset otherwise

HALT

The HALT instruction suspends the central processor operation until a subsequent interrupt or reset is received. While in the halt state, the processor will execute NOPs to maintain memory refresh logic.

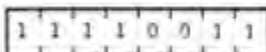


Cycles: 1
States: 4
Flags: none

DI

IFF = 0

DI disables the maskable interrupt by resetting the interrupt enable flip-flops (IFF1 and IFF2). Note: this instruction disables the maskable interrupt during its execution.

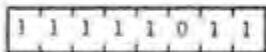


Cycles: 1
States: 4
Flags: none

EI

IFF = 1

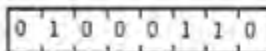
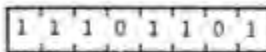
EI enables the maskable interrupt by setting the interrupt enable flip-flops (IFF1 and IFF2). Note: this instruction disables the maskable interrupt during its execution.



Cycles: 1
States: 4
Flags: none

IM 0

The IM 0 instruction sets interrupt mode 0. In this mode the interrupting device can insert any instruction on the data bus and allow the central processor to execute it.



Cycles: 2
States: 8
Flags: none

IM 1

The IM 1 instruction sets interrupt mode 1. In this mode the processor will respond to an interrupt by executing a restart of location 0038H.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

Cycles: 2
States: 8
Flags: none

0	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---

IM 2

The IM 2 instruction sets interrupt mode 2. This mode allows an indirect call to any location in memory. With this mode, the central processor forms a 16-bit memory address. The upper 8 bits are the contents of the Interrupt Vector Register I and the lower 8 bits are supplied by the interrupting device.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

Cycles: 2
States: 8
Flags: none

0	1	0	1	1	1	1	0
---	---	---	---	---	---	---	---

SIXTEEN-BIT ARITHMETIC GROUP

ADD HL, ss

$HL \leftarrow HL + ss$

The contents of register pair ss are added to the contents of register pair HL and the result is stored in HL.

0	0	ss	ss	1	0	0	1
---	---	----	----	---	---	---	---

Cycles: 3
States: 11
Flags: H, N, C

H: set if carry out of bit 11; reset otherwise
N: reset
C: set if carry from bit 15; reset otherwise

ADC HL, ss

$HL \leftarrow HL + ss + CY$

The contents of register pair ss are added with the Carry Flag to the contents of the register pair HL, and the result is stored in HL.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

Cycles: 4
States: 15
Flags: S, Z, H, N, C, P/V

0	1	ss	ss	1	0	1	0
---	---	----	----	---	---	---	---

S: set if result is negative; reset otherwise
Z: set if result is 0; reset otherwise
H: set if carry out of bit 11; reset otherwise
N: reset
C: set if carry from bit 15; reset otherwise
P/V: set if overflow; reset otherwise

SBC HL, ss**HL** ← **HL** − **ss** − **CY**The contents of the register pair **ss** and the Carry Flag are subtracted from the contents of register pair **HL**, and the result is stored in **HL**.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

0	1	s	s	0	0	1	0
---	---	---	---	---	---	---	---

Cycles: 4

States: 15

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

Z: set if result is 0; reset otherwise

H: set if no borrow from bit 12; reset otherwise

N: set

C: set if no borrow; reset otherwise

P/V: set if overflow; reset otherwise

ADD IX, pp**IX** ← **IX** + **pp**The contents of register pair **pp** are added to the contents of the Index Register **IX**, and the results are stored in **IX**.

1	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---

0	0	p	p	1	0	0	1
---	---	---	---	---	---	---	---

Cycles: 4

States: 15

Flags: H, N, C

H: set if carry out of bit 11; reset otherwise

N: reset

C: set if carry from bit 15; reset otherwise

ADD IY, rr**IY** ← **IY** + **rr**The contents of register pair **rr** are added to the contents of Index Register **IY**, and the result is stored in **IY**.

1	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

0	0	r	r	1	0	0	1
---	---	---	---	---	---	---	---

Cycles: 4

States: 15

Flags: H, N, C

H: set if carry out of bit 11; reset otherwise

N: reset

C: set if carry from bit 15; reset otherwise

INC ss**ss** ← **ss** + 1The contents of register pair **ss** are incremented.

0	0	s	s	0	0	1	1
---	---	---	---	---	---	---	---

Cycles: 1
States: 6
Flags: none

INC IX

$IX \leftarrow IX + 1$

The contents of the Index Register IX are incremented.

1	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

Cycles: 2
States: 10
Flags: none

INC IY

$IY \leftarrow IY + 1$

The contents of the Index Register IY are incremented.

1	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

Cycles: 2
States: 10
Flags: none

DEC ss

$ss \leftarrow ss - 1$

The contents of register pair ss are decremented.

0	0	s	s	1	0	1	1
---	---	---	---	---	---	---	---

Cycles: 1
States: 6
Flags: none

DEC IX

$IX \leftarrow IX - 1$

The contents of the Index Register IX are decremented.

1	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---

0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Cycles: 2
States: 10
Flags: none

DEC IY

$IY \leftarrow IY - 1$

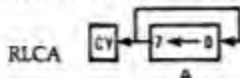
The contents of the Index Register IY are decremented.

1	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

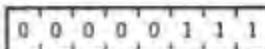
0	0	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Cycles: 2
States: 10
Flags: none

ROTATE AND SHIFT GROUP



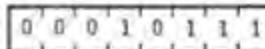
The contents of the Accumulator are rotated left. The content of bit 7 is copied into the Carry Flag, and also into bit 0.



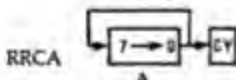
Cycles: 1
States: 4
Flags: H, N, C
H: reset
N: reset
C: data from bit 7 of Accumulator



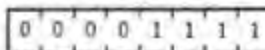
The contents of the Accumulator are rotated left. The content of bit 7 is copied into the Carry Flag, and the previous content of the Carry Flag is copied into bit 0.



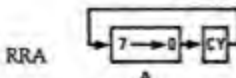
Cycles: 1
States: 4
Flags: H, N, C
H: reset
N: reset
C: data from bit 7 of Accumulator.



The contents of the Accumulator are rotated right. The content of bit 0 is copied into bit 7 and also into the Carry Flag.

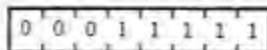


Cycles: 1
States: 4
Flags: H, N, C
H: reset
N: reset
C: data from bit 0 of Accumulator.



The contents of the Accumulator are rotated right. The content of bit 0 is copied into the Carry Flag, and the previous content of the Carry Flag is

copied into bit 7.



Cycles: 1

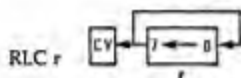
States: 4

Flags: H, N, C

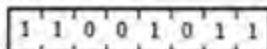
H: reset

N: reset

C: data from bit 0 of Accumulator.



The 8-bit contents of register *r* are rotated left. The content of bit 7 is copied into the Carry Flag and also into bit 0.



Cycles: 2

States: 8

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

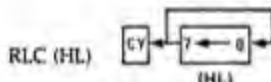
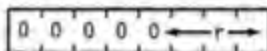
Z: set if result is 0; reset otherwise

H: reset

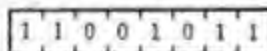
N: reset

C: data from bit 7 of source register

P/V: set if parity even; reset otherwise



The contents of the memory address specified by the contents of register pair HL are rotated left. The content of bit 7 is copied into the Carry Flag and also into bit 0.



Cycles: 4

States: 15

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

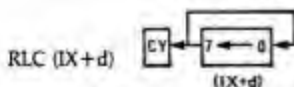
Z: set if result is 0; reset otherwise

H: reset

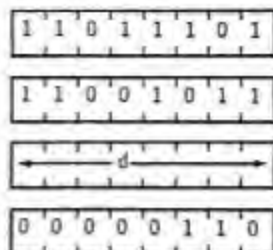
N: reset

C: data from bit 7 of source register

P/V: set if parity even; reset otherwise



The contents of the memory address, specified by the sum of the contents of the Index Register IX and a two's complement displacement integer *d*, are rotated left. The content of bit 7 is copied into the Carry Flag and also into bit 0.



Cycles: 6

States: 23

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

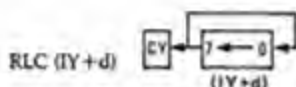
Z: set if result is 0; reset otherwise

H: reset

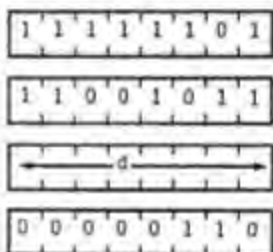
N: reset

C: data from bit 7 of source register

P/V: set if parity even; reset otherwise



The contents of the memory address, specified by the sum of the contents of the Index Register IX and a two's complement displacement integer d , are rotated left. The content of bit 7 is copied into the Carry Flag and also into bit 0.



Cycles: 6

States: 23

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

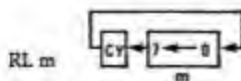
Z: set if result is 0; reset otherwise

H: reset

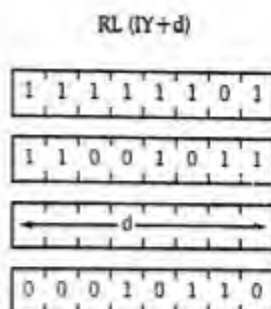
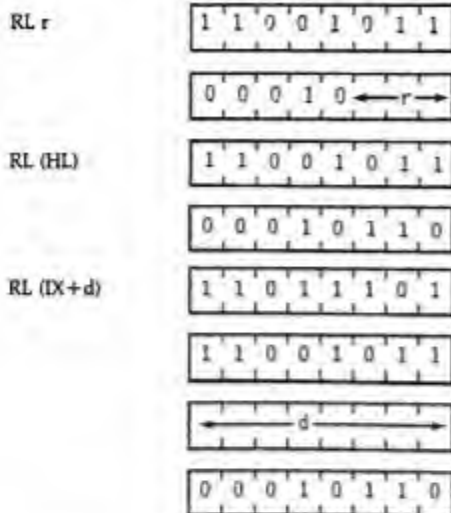
N: reset

C: data from bit 7 of source register

P/V: set if parity even; reset otherwise



The contents of the m operand are rotated left. The content of bit 7 is copied into the Carry Flag and the previous content of the Carry Flag is copied into bit 0.



Instruction	Cycles	States
RL r	2	8
RL (HL)	4	15
RL (IX+d)	6	23
RL (IY+d)	6	23

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

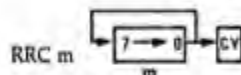
Z: set if result is 0; reset otherwise

H: reset

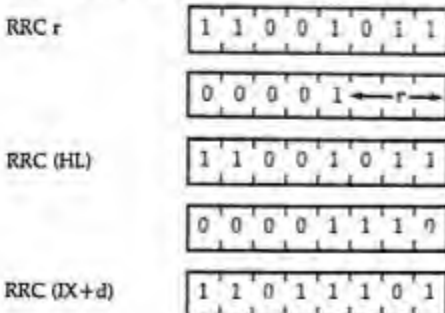
N: reset

C: data from bit 7 of source register

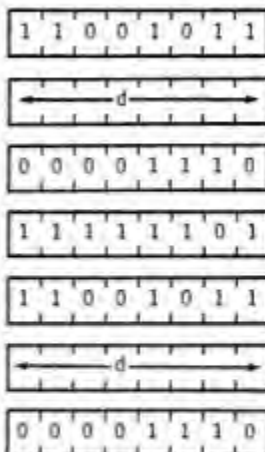
P/V: set if parity even; reset otherwise



The contents of the operand m are rotated right. The content of bit 0 is copied into the Carry Flag and also into bit 7.



RRC (IY+d)



Instruction	Cycles	States
RRC r	2	8
RRC (HL)	4	15
RRC (IX+d)	6	23
RRC (IY+d)	6	23

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

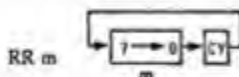
Z: set if result is 0; reset otherwise

H: reset

N: reset

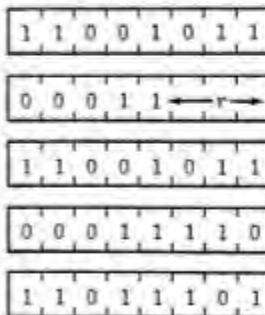
C: data from bit 0 of source register

P/V: set if parity even; reset otherwise



The contents of operand *m* are rotated right. The content of bit 0 is copied into the Carry Flag, and the previous content of the Carry Flag is copied into bit 7.

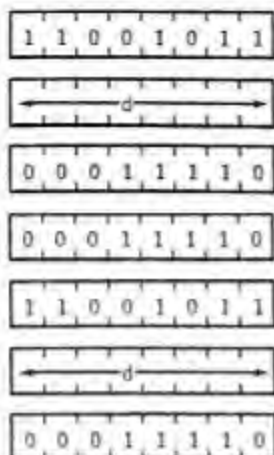
RR r



RR (HL)

RR (IX+d)

RR (IY+d)



Instruction	Cycles	States
RR r	2	8
RR (HL)	4	15
RR (IX+d)	6	23
RR (IY+d)	6	23

Flags: S, Z, H, N, C, P/V

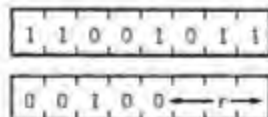
S: set if result is negative; reset otherwise
 Z: set if result is 0; reset otherwise
 H: reset
 N: reset
 C: data from bit 0 of source register
 P/V: set if parity even; reset otherwise

SLA m

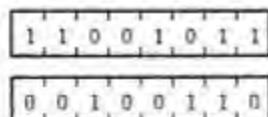


An arithmetic shift left is performed on the contents of operand m. Bit 0 is reset. The content of bit 7 is copied into the Carry Flag.

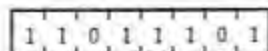
SLA r



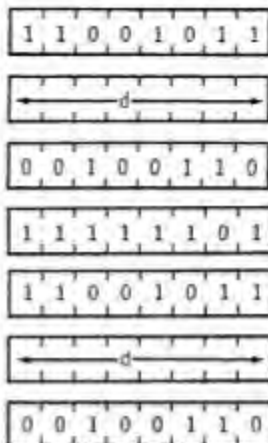
SLA (HL)



SLA (IX+d)



SLA (Y+d)



Instruction	Cycles	States
SLA r	2	8
SLA (HL)	4	15
SLA (IX+d)	6	23
SLA (IY+d)	6	23

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

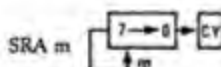
Z: set if result is 0; reset otherwise

H: reset

N: reset

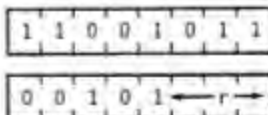
C: data from bit 7

P/V: set if parity even; reset otherwise

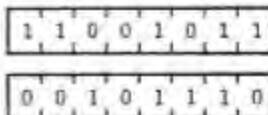


An arithmetic shift right is performed on the contents of operand *m*. The content of bit 0 is copied into the Carry Flag, and the previous content of bit 7 is unchanged.

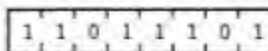
SRA r



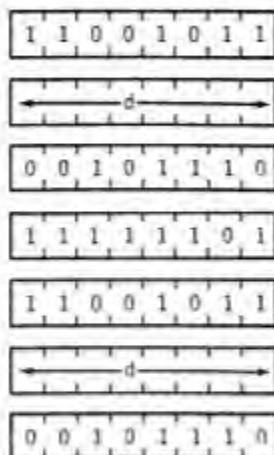
SRA (HL)



SRA (IX+d)



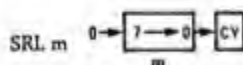
SRA (IX+d)



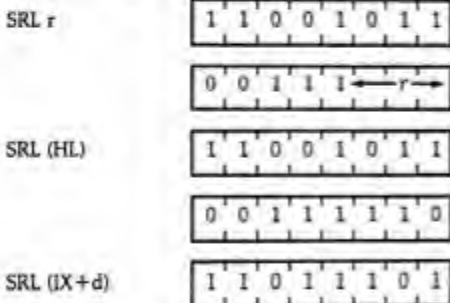
Instruction	Cycles	States
SRA r	2	8
SRA (HL)	4	15
SRA (IX+d)	6	23
SRA (IY+d)	6	23

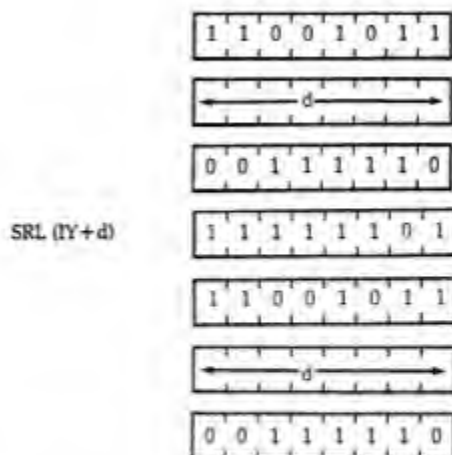
Flags: S, Z, H, N, C, P/V

- S: set if result is negative; reset otherwise
- Z: set if result is 0; reset otherwise
- H: reset
- N: reset
- C: data from bit 0 of source register
- P/V: set if parity even; reset otherwise



The contents of operand m are shifted right. The content of bit 0 is copied into the Carry Flag and bit 7 is reset.





Instruction	Cycles	States
SRL r	2	8
SRL (HL)	4	15
SRL (IX+d)	6	23
SRL (IY+d)	6	23

Flags: S, Z, H, N, C, P/V

S: set if result is negative; reset otherwise

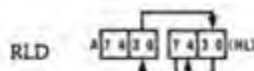
Z: set if result is 0; reset otherwise

H: reset

N: reset

C: data from bit 0 of source register

P/V: set if parity even; reset otherwise



The contents of the low-order 4 bits of memory location (HL) are copied into the high-order 4 bits of that same memory location. The previous contents of those high-order 4 bits are copied into the low-order 4 bits of the Accumulator, and the previous contents of the low-order 4 bits of the Accumulator are copied into the low-order 4 bits of the memory location (HL). The contents of the high-order 4 bits of the Accumulator are unaffected.



Cycles: 5

States: 18

Flags: S, Z, H, N, P/V

S: set if Accumulator is negative after operation; reset otherwise

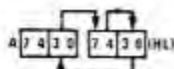
Z: set if Accumulator is 0 after operation; reset otherwise

H: reset

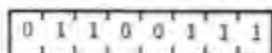
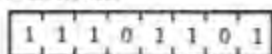
N: reset

P/V: set if parity of Accumulator is even after operation; reset otherwise

RRD



The contents of the low-order 4 bits of memory location (HL) are copied into the low-order 4 bits of the Accumulator. The previous contents of the low-order 4 bits of the Accumulator are copied into the high-order 4 bits of location (HL), and the previous contents of the high-order 4 bits of (HL) are copied into the low-order 4 bits of (HL). The contents of the high-order 4 bits of the Accumulator are unaffected.



Cycles: 5

States: 18

Flags: S, Z, H, N, P/V

S: set if Accumulator is negative after operation; reset otherwise

Z: set if Accumulator is 0 after operation; reset otherwise

H: reset

N: reset

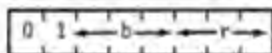
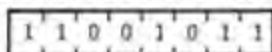
P/V: set if parity of Accumulator is even after operation; reset otherwise

BIT SET, RESET AND TEST GROUP

BIT b, r

 $Z \leftarrow \overline{r}$

After execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the indicated register.



Cycles: 2

States: 8

Flags: S, Z, H, N, P/V

S: unknown

Z: set if specified bit is 0; reset otherwise

H: set

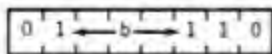
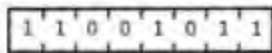
N: reset

P/V: unknown

BIT b, (HL)

 $Z \leftarrow \overline{(HL)_b}$

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the HL register pair.



Cycles: 3

States: 12

Flags: S, Z, H, N, P/V

S: unknown

Z: set if specified bit is 0; reset otherwise

H: set

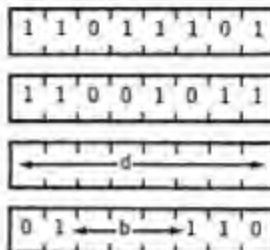
N: reset

P/V: unknown

BIT b, (IX+d)

$$Z = \overline{(IX+d)_b}$$

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents of register pair IX and the two's complement displacement integer d.



Cycles: 5
States: 20
Flags: S, Z, H, N, P/V

S: unknown

Z: set if specified bit is 0; reset otherwise

H: set

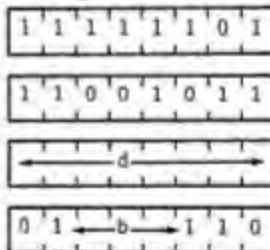
N: reset

P/V: unknown

BIT b, (IY+d)

$$Z = \overline{(IY+d)_b}$$

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents of register pair IY and the two's complement displacement integer d.



Cycles: 5
States: 20
Flags: S, Z, H, N, P/V

S: unknown

Z: set if specified bit is 0; reset otherwise

H: set

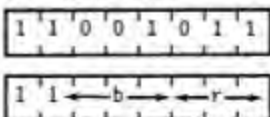
N: reset

P/V: unknown

SET b, r

$$r_b \leftarrow 1$$

Bit b (any bit, 7 thru 0) in register r is set.

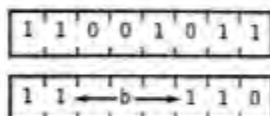


Cycles: 2
States: 8
Flags: none

SET b, (HL)

$(HL)_8 \leftarrow 1$

Bit b in the memory location addressed by the contents of register pair HL is set.

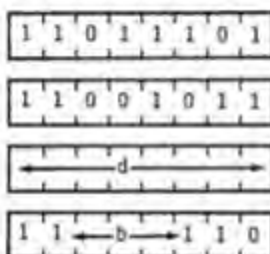


Cycles: 4
States: 15
Flags: none

SET b, (IX+d)

$(IX+d)_8 \leftarrow 1$

Bit b in the memory location addressed by the sum of the contents of the IX register pair and the two's complement displacement integer d is set.

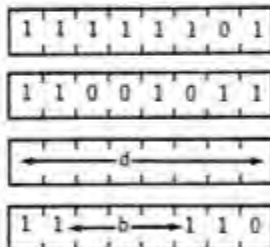


Cycles: 6
States: 23
Flags: none

SET b, (IY+d)

$(IY+d)_8 \leftarrow 1$

Bit b in the memory location addressed by the sum of the contents of the IY register pair and the two's complement displacement integer d is set.



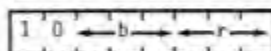
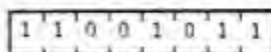
Cycles: 6
States: 23
Flags: none

RES b, m

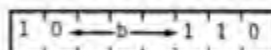
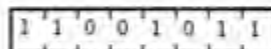
$s_6 = 0$

Bit b in operand m is reset.

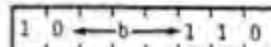
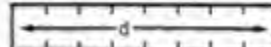
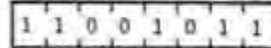
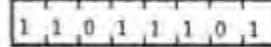
RES b, r



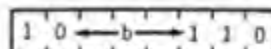
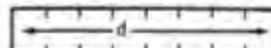
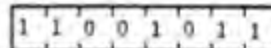
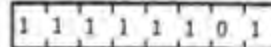
RES b, (HL)



RES b, (IX+d)



RES b, (IY+d)



Instruction

Cycles

States

RES b, r

4

8

RES b, (HL)

4

15

RES b, (IX+d)

6

23

RES b, (IY+d)

6

23

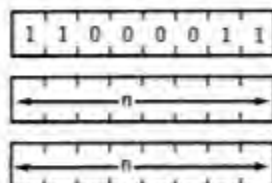
Flags: none

JUMP GROUP

JP nn

PC ← nn

Operand nn is loaded into register pair PC (program counter) and points to the address of the next program instruction to be executed.



Cycles: 3
States: 10
Flags: none

JP cc, nn

IF cc TRUE, PC ← nn

If condition cc is true, the instruction loads operand nn into register pair PC, and the program continues with the instruction beginning at address nn. If condition cc is false, the program counter is incremented as usual, and the program continues with the next sequential instruction.

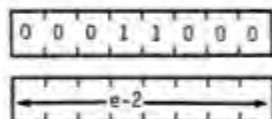


Cycles: 3
States: 10
Flags: none

JR e

PC ← PC + e

This instruction provides for unconditional branching to other segments of a program. The value of the displacement e is added to the PC and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes.



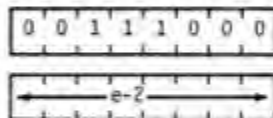
Cycles: 3
States: 12
Flags: none

JR C, e

If C=0, continue

If C=1, $PC \leftarrow PC + e$

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is set, the value of the displacement e is added to the PC, and the next instruction is fetched from the location designated by the new contents of the PC. If the flag is reset the next instruction is taken from the location following this instruction.



If the condition is met:

Cycles: 3

States: 12

If the condition is not met:

Cycles: 2

States: 7

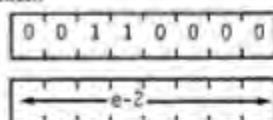
Flags: none

JR NC, e

If C=1, continue

If C=0, $PC \leftarrow PC + e$

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is reset, the value of the displacement e is added to the PC, and the next instruction is fetched from the location designated by the new contents of the PC. If the flag is set, the next instruction to be executed is taken from the location following this instruction.



If the condition is met:

Cycles: 3

States: 12

If the condition is not met:

Cycles: 2

States: 7

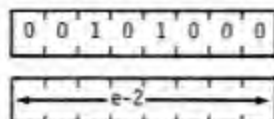
Flags: none

JR Z, e

If Z=0, continue

If Z=1, $PC \leftarrow PC + e$

If the Zero Flag is set, the value of the displacement e is added to the PC and the next instruction is fetched from the location designated by the new contents of the PC. If the Zero Flag is reset, the next instruction to be executed is taken from the location following this instruction.



If the condition is met:

Cycles: 3

States: 12

If the condition is not met:

Cycles: 2

States: 7

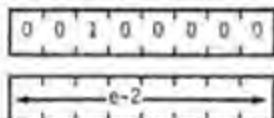
Flags: none

JR NZ, e

If Z=1, continue

If Z=0, PC ← PC+e

If the Zero Flag is reset, the value of the displacement e is added to the PC, and the next instruction is fetched from the location designated by the new contents of the PC. If the Zero Flag is set, the next instruction to be executed is taken from the location following this instruction.



If the condition is met:

Cycles: 3

States: 12

If the condition is not met:

Cycles: 2

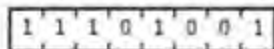
States: 7

Flags: none

JP (HL)

PC ← HL

The PC is loaded with the contents of the HL register pair. The next instruction is fetched from the location designated by the new contents of the PC.



Cycles: 1

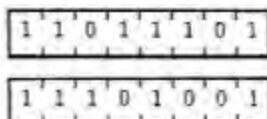
States: 4

Flags: none

JP (IX)

PC ← IX

The PC is loaded with the contents of the IX Register Pair. The next instruction is fetched from the location designated by the new contents of the PC.

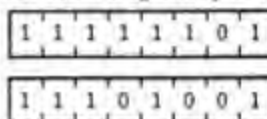


Cycles: 2
States: 8
Flags: none

JP (IY)

PC ← IY

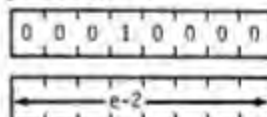
The PC is loaded with the contents of the IY Register Pair. The next instruction is fetched from the location designated by the new contents of the PC.



Cycles: 2
States: 8
Flags: none

DJNZ, e

The B register is decremented, and if a non 0 value remains, the value of the displacement e is added to the PC. The next instruction is fetched from the location designated by the new contents of the PC. If the result of decrementing leaves B with a 0 value, the next instruction to be executed is taken from the location following this instruction.



If B ≠ 0:
Cycles: 3
States: 13

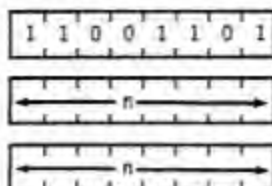
If B = 0:
Cycles: 2
States: 8
Flags: none

CALL AND RETURN GROUP

CALL nn

$(SP-1) \leftarrow PC_{old}$, $(SP-2) \leftarrow PC_{old}$, $PC \leftarrow nn$

After pushing the current contents of the PC onto the top of the external memory stack, the operands nn are loaded into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. Note: because this is a 3-byte instruction, the PC will have been incremented by three before the push is executed.

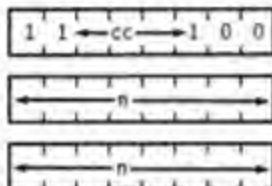


Cycles: 5
States: 17
Flags: none

CALL cc, nn

If cc TRUE: $(SP-1) \leftarrow PC_n$, $(SP-2) \leftarrow PC_{n+1}$, $PC \leftarrow nn$

If condition cc is true, this instruction pushes the current contents of the PC onto the top of the external memory stack, then loads the operands nn into PC to point to the address in memory where the first opcode of a subroutine is to be fetched.



If cc is true:

Cycles: 5
States: 17

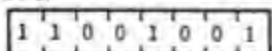
If cc is false:

Cycles: 3
States: 10
Flags: none

RET

$PC_n \leftarrow (SP)$, $PC_{n+1} \leftarrow (SP+1)$

Control is returned to the original program flow by popping the previous contents of the PC off the top of the external memory stack, where they were pushed by the CALL instruction. On the following machine cycle, the central processor will fetch the next program opcode from the location in memory now pointed to by the PC.



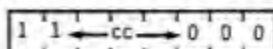
Cycles: 3
States: 10
Flags: none

RET cc

If cc TRUE: $PC_n \leftarrow (SP)$, $PC_{n+1} \leftarrow (SP+1)$

If condition cc is true, control is returned to the original program flow by popping the previous contents of the PC off the top of the external memory stack where they were pushed by the CALL instruction. On the following machine cycle, the central processor will fetch the next program opcode from

the location in memory now pointed to by the PC. If condition cc is false, the PC is simply incremented as usual, and the program continues with the next sequential instruction.



If cc is true:

Cycles: 3
States: 11

If cc is false:

Cycles: 1
States: 5
Flags: none

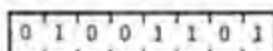
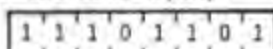
RETI

Return from interrupt

This instruction is used at the end of an interrupt service routine to

1. Restore the contents of the PC.
2. Signal an I/O device that the interrupt routine has been completed.

The RETI instruction facilitates the nesting of interrupts allowing higher priority devices to suspend service of lower priority service routines. This instruction also resets the IFF1 and IFF2 flip-flops.

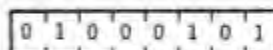
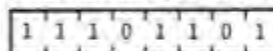


Cycles: 4
States: 14
Flags: none

RETN

Return from nonmaskable interrupt

Used at the end of a service routine for a nonmaskable interrupt, the instruction executes an unconditional return which functions identically to the RET instruction. Control is now returned to the original program flow; on the following machine cycle the central processor will fetch the next opcode from the location in memory now pointed to by the PC. Also, the state of IFF2 is copied back into IFF1 to the state it had prior to the acceptance of the NMI.



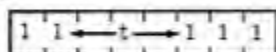
Cycles: 4
States: 14
Flags: none

RST p

$(SP-1) \leftarrow PC_n$, $(SP-2) \leftarrow PC_{n-1}$, $PC_n \leftarrow 0$, $PC_1 \leftarrow p$

The current PC contents are pushed onto the external memory stack, and the

page zero memory location given by operand p is loaded into the PC. Program execution then begins with the opcode in the address now pointed to by PC. The restart instruction allows for a jump to one of 8 addresses as shown in the table below. The operand p is assembled into the object code using the corresponding t state.



p	t
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

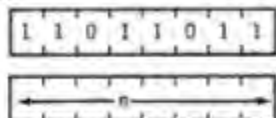
Cycles: 3
States: 11
Flags: none

INPUT AND OUTPUT GROUP

IN $A, (n)$

$A \leftarrow (n)$

The operand n is placed on the bottom half of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half of the address bus at this time. One byte from the selected port is then placed on the data bus and written into the Accumulator in the central processor.

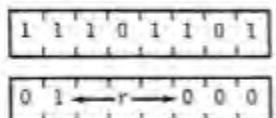


Cycles: 3
States: 11
Flags: none

IN $r, (C)$

$r \leftarrow (C)$

The contents of register C are placed on the bottom half of the address bus to select the I/O device at one of 256 possible ports. The contents of register B are placed on the top half of the address bus at this time. One byte from the selected port is then placed on the data bus and written into register r in the central processor.



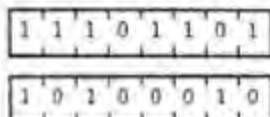
Cycles: 3
States: 12
Flags: S, Z, H, N, P/V

S: set if input data is negative; reset otherwise
 Z: set if input data is 0; reset otherwise
 H: reset
 N: reset
 P/V: set if parity is even; reset otherwise

INI

(HL) ← (C), B ← B-1, HL ← HL+1

The contents of register C are placed on the bottom half of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half of the address bus. One byte from the selected port is then placed on the data bus and written to the central processor. The contents of the HL register pair are then placed on the address bus, and the input byte is written into the corresponding location of memory. Finally, the byte counter is decremented, and register pair HL is decremented.



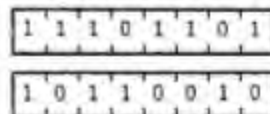
Cycles: 4
 States: 16
 Flags: S, Z, H, N, P/V

S: unknown
 Z: set if B-1=0; reset otherwise
 H: unknown
 N: set
 P/V: unknown

INIR

(HL) ← (C), B ← B-1, HL ← HL+1

The contents of register C are placed on the bottom half of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half of the address bus. One byte is selected and is placed on the data bus and written into the central processor. The contents of the HL register pair are placed on the address, and the input byte is written into the corresponding memory location. The byte counter is then decremented and the HL register pair is incremented. If decrementing causes B to go to 0, the instruction is terminated. If B is not 0, the PC is decremented by two and the instruction repeated. Interrupts will be recognized after each data transfer.



If B ≠ 0:

Cycles: 5
 States: 21

If B = 0:

Cycles: 4
 States: 16

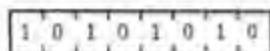
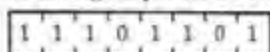
Flags: S, Z, H, N, P/V

S: unknown
 Z: set
 H: unknown
 N: set
 P/V: unknown

IND

(HL) ← (C), B ← B-1, HL ← HL-1

The contents of register C are placed on the bottom half of the address bus to select the I/O device. Register B may be used as a byte counter, and its contents are placed on the top half of the address bus. One byte from the selected port is placed on the data bus and written to the central processor. The contents of the HL register pair are placed on the address bus, and the input byte is written into the corresponding memory location. Finally, the byte counter and register pair HL are decremented.



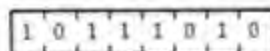
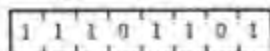
Cycles: 4
 States: 16
 Flags: S, Z, H, N, P/V

S: unknown
 Z: set if B-1=0; reset otherwise
 H: unknown
 N: set
 P/V: unknown

INDR

(HL) ← (C), B ← B-1, HL ← HL-1

The contents of register C are placed on the bottom half of the address bus to select the I/O device. Register B is used as a byte counter, and its contents are placed on the top half of the address bus. One byte from the selected port is placed on the data bus and written to the central processor. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding memory location. The HL register pair and the byte counter are then decremented. If decrementing causes B to go to 0, the instruction is terminated. If B is not 0, the PC is decremented by 2, and the instruction is repeated. Interrupts will be recognized after each data transfer.



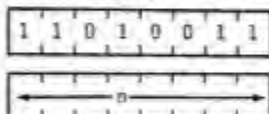
If B ≠ 0:
 Cycles: 5
 States: 21

If B = 0:
 Cycles: 4
 States: 16
 Flags: S, Z, H, N, P/V

S: unknown
 Z: set
 H: unknown
 N: set
 P/V: unknown

OUT (n), A
(n) ← A

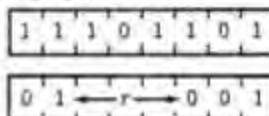
The operand *n* is placed on the bottom half of the address bus to select the I/O device. The contents of the Accumulator appear on the top half of the address bus. Then the byte contained in the Accumulator is placed on the data bus and written into the selected peripheral device.



Cycles: 3
States: 11
Flags: none

OUT (C), r
(C) ← r

The contents of register C are placed on the bottom half of the address bus to select the I/O device. The contents of register B are placed on the top half of the address bus. The byte contained in register *r* is placed on the data bus and written into the selected peripheral device.

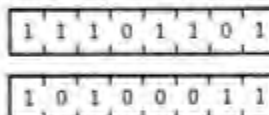


Cycles: 3
States: 12
Flags: none

OUTI

(C) ← (HL), B ← B-1, HL ← HL+1

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the central processor. After the byte counter (B) is decremented, the contents of register C are placed on the bottom half of the address bus to select the I/O device. Register B may be used as a byte counter, and its decremented value is placed on the top half of the address bus. The byte to be output is placed on the data bus and written into the selected peripheral device. Finally, the register pair HL is incremented.



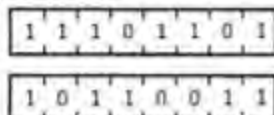
Cycles: 4
States: 16
Flags: S, Z, H, N, P/V

S: unknown
Z: set if B-1=0; reset otherwise
H: unknown
N: set
P/V: unknown

OTIR

(C) ← (HL), B ← B-1, HL ← HL+1

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the central processor. After the byte counter (B) is decremented, the contents of register C are placed on the bottom half of the address bus to select the I/O device. Register B may be used as a byte counter, and its decremented value is placed on the top half of the address bus at this time. The byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is incremented. If the decremented B register is not 0, the PC is decremented by two and the instruction is repeated. If B is 0, the instruction is terminated. Interrupts will be recognized after each data transfer.



If B ≠ 0:

Cycles: 5

States: 21

If B = 0:

Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: unknown

Z: set

H: unknown

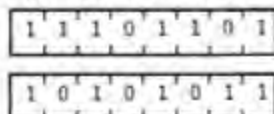
N: set

P/V: unknown

OUTD

(C) ← (HL), B ← B-1, HL ← HL-1

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the central processor. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half of the address bus to select the I/O device. Register B may be used as a byte counter, and its decremented value is placed on the top half of the address bus. The byte to be output is placed on the data bus written into the selected peripheral device. Finally, the register pair HL is decremented.



Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: unknown

Z: set if B-1=0; reset otherwise

H: unknown

N: set

P/V: unknown

OTDR

(C) ← (HL), B ← B-1, HL ← HL-1

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in central processors. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half of the address bus to select the I/O device. Register B may be used as a byte counter, and its decremented value is placed on the top half of the address bus. The byte to be output is then placed on the data bus and written into the selected peripheral device. Register pair HL is then decremented. If the decremented B register is not 0, the PC is decremented by 2, and the instruction is repeated. If register B is 0, then the instruction is terminated. Interrupts will be recognized after each data transfer.

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

1	0	1	1	1	0	1	1
---	---	---	---	---	---	---	---

If B ≠ 0:

Cycles: 5

States: 21

If B = 0:

Cycles: 4

States: 16

Flags: S, Z, H, N, P/V

S: unknown

Z: set

H: unknown

N: set

P/V: unknown

CHAPTER 4

BUILD YOUR OWN COMPUTER—Start With the Basics

The computer to be built from the design described in this book is called ZAP, for Z80 Applications Processor. Building a computer from scratch is both educational and utilitarian (and it saves money). I explain each section of the construction process in detail. Ideally, each step should be tested before proceeding on to the next stage. While this is not possible in all cases, there is a beneficial side effect in taking this route. Often good designs fail to work because the level of construction is beyond the ability of the builder.

I've made the assumption that most hobbyists do not possess sophisticated test equipment, such as oscilloscopes or logic analyzers, and as a result, I've kept testing procedures as simple as possible. By dividing ZAP into logical milestones for checkout and test (and using proven components), problems can be identified at earlier stages and rectified more easily.

The initial implementation of ZAP will constitute a minimum operable configuration. It is important that this works before you attempt to add any of the optional peripherals. Every effort will be made to familiarize the reader with the components of each section and the philosophy of design. While it is necessary to assemble all the components of this minimum configuration completely in order to check proper central processor operation, comprehensive subassembly pretesting should (I hope) correct any wiring errors.

The basic ZAP is divided into four major subassemblies: Z80 busing and control, memory and I/O chip select decoding, memory, and input/output registers. These major divisions are further divided at the component level. Schematics include a complete explanation of their logical function, and test procedures are outlined after each construction presentation.

The Processor

Figure 4.1 is a detailed block diagram of the basic ZAP computer.

I. Z80 Busing and Control Logic

A. Clock Generation

The ZAP computer runs on a 2.5 MHz TTL clock. Unlike the 8080A, the Z80 requires only a single-phase clock and can be driven from DC to 2.5 MHz (the 8080A runs to 4 MHz). Figure 4.2 illustrates the basic timing cycle of the computer.

Each basic operation (M_n) of the computer is completed in three or six clock periods. Figure 4.2 shows a typical instruction cycle which consists of three machine cycles: fetch, memory read, and memory write. After the opcode of the instruction is fetched during M1, the subsequent cycles move the data between memory and the central processor.

Figures 4.3a and 4.3b illustrate two possible clock designs for the Z80. Both clock circuits have a 330 ohm pull-up to +5 V. This will satisfy both the AC and DC clock signal requirements, but it is best to use a separate inverter gate

section to drive the pull-up whatever the oscillation technique.

The crystal controlled circuit of figure 4.3a is preferred if consistent execution time is to be maintained. Thus, the circuit of figure 4.3b, though otherwise acceptable, should be avoided if the computer is to be used as an event timer. It can serve a very useful purpose in the development stages, however, by allowing the user to slow the clock down (by increasing the values of R and C) to a rate where it is possible to directly monitor the central processor operation. Should it ever be necessary to single-step the clock, the circuit in figure 4.4 should be used. Given the multiple clock cycles necessary to execute a single instruction, it would take a lot of button pushes to follow a program through execution.

A much easier diagnostic method would be to use an instruction single-stepping circuit. The circuit, shown in figure 4.5, is not part of the finished schematic of ZAP because it is necessary only if the builder has a problem and needs to follow the execution of a program instruction by instruction. This single-stepping function is accomplished by using the control signals generated by the Z80 during program execution. The two particular signals of concern are \overline{MI} and \overline{WAIT} . \overline{MI} is an output, and \overline{WAIT} is an input. As shown in figure 4.6, \overline{MI} goes to a logic 0 level at the beginning of every instruction fetch cycle. \overline{MI} signifies that the computer has completed one instruction and is starting on the next. The objective is to stop the microprocessor before it executes this next instruction.

The \overline{WAIT} input to the Z80 does just that. A logic 0 level applied to this input will suspend the program execution of the computer and indefinitely hold it in the \overline{MI} cycle. During T_3 , the central processor samples the \overline{WAIT} input line with the trailing edge of the clock. If, at this time, \overline{WAIT} is at a logic 0 level, an additional wait state will be entered, and the line will be sampled again. The central processor will hang in this mode until \overline{WAIT} is raised to a logic 1. It should be noted that this is not a computer halt command.

The real purpose behind these signals is to allow the relatively slow memory and peripherals to be used with a very fast central processor. Extra wait states should be inserted only when necessary for the central processor to access these devices. The effect is to synchronize the timing between the central processor and its I/O devices. The circuit of figure 4.5 allows us to control the \overline{WAIT} state and to execute only one instruction with each press of the button. The output at IC 1, pin 8 (the \overline{WAIT} input) is normally low, causing an indefinite wait. When the button is pushed, a single debounced pulse clocks IC 2, which is a D-type flip-flop. The duration of this pulse (the time you hold the button down) is irrelevant, because the flip-flop is edge triggered and is only concerned with the leading edge. Pressing the button sets IC 2 and raises the \overline{WAIT} line. No longer told to wait, the central processor executes the instruction at full clock speed. As it is about to start the next instruction fetch cycle, \overline{MI} goes low as before, and triggers the one-shot. When it fires, IC 3 resets IC 2 and returns the central processor to a wait condition until the next time the button is pushed.

The single-step feature isn't of much use in a computer unless there is some way to monitor the contents of all the registers and to determine what the computer is trying to do at any one time. To accomplish this, ZAP must be completely operational and be running a breakpoint-monitor program which allows the user to single-step with a software routine. We'll discuss such programs later.

This fact is of small consolation to a person with a partially debugged computer or hardware error that keeps side-tracking large programs. While it would be nice to see all the register contents, it is virtually impossible to do so without having a central processor that can run a dump and display routine. This cannot be done using the hardware stepping circuit of figure 4.5. It is possible, however, to look at the contents of the address and data buses while the central processor is stopped. This should give a good indication as to

whether the computer is operating properly.

Many instruments can be used to read the TTL levels on the buses. A scope or high-impedance voltmeter can be used, but a visible display of the bus contents is a better idea. The circuits in figure 4.7 show simple methods to display the contents of the address and data buses. The circuits are included as aids and are not necessary for the operation of ZAP.

Basically, the circuit of figure 4.7a is a simple LED driver that is duplicated 16 times for the address bus and 8 times for the data bus. Because the Z80 should drive only one TTL load from each output pin (bus driver inputs are already attached), any display drivers of this type must be attached on the output side of the bus drivers. This circuit will serve as a rudimentary front panel for any builders who feel a computer isn't complete without flashing lights.

Sometimes the need arises to monitor a single point in a circuit and watch for level changes. While the LED driver of figure 4.7a would detect a slowly changing level, it would miss short pulses such as M1. To monitor the occurrence of such events, especially if no oscilloscope is available for testing purposes, it is advisable to build the circuit in figure 4.7b. This simple logic probe is adequate for most applications, but care must be taken in its use. It cannot detect an open circuit and the pulse detector only triggers on the negative edge of any transition. Should that present any problems, add the optional circuit using the 7486; that will allow it to detect either edge.

The logic probe or similar logic level detector (scope, DVM, VOM, etc.) is necessary to statically test the subassemblies.

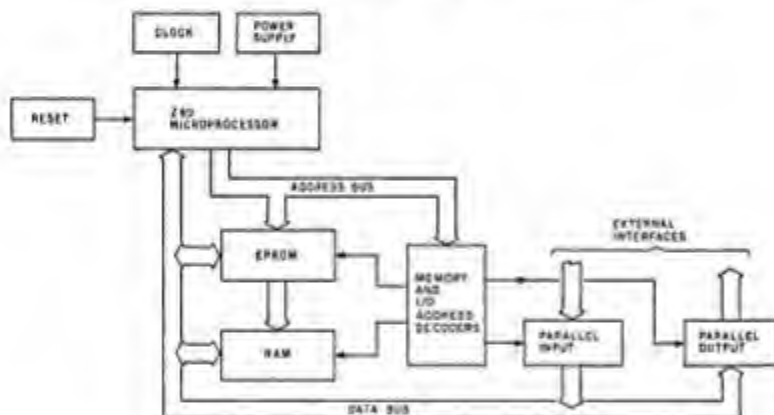


Figure 4.1 A block diagram of a minimum ZAP system.

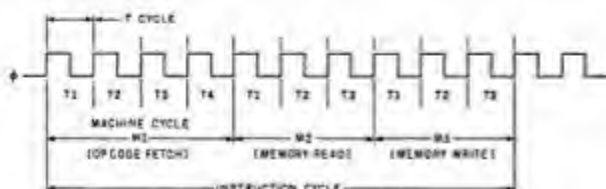


Figure 4.2 An example of timing during a typical instruction cycle.

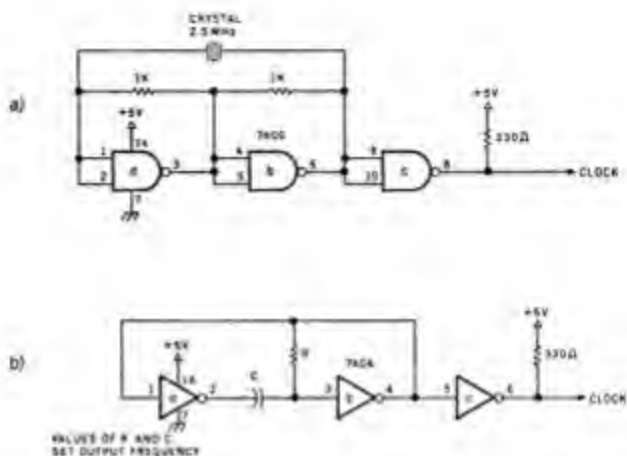


Figure 4.3 Typical 2.5 MHz clock circuits for the Z80.

- a) With crystal control.
b) With a variable-frequency oscillator.

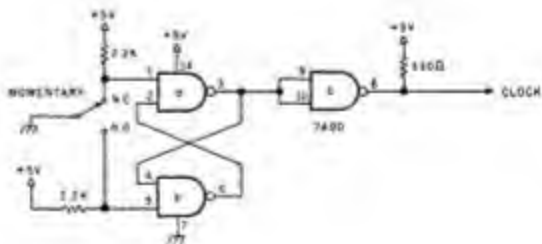


Figure 4.4 A single-cycle clock generator circuit.

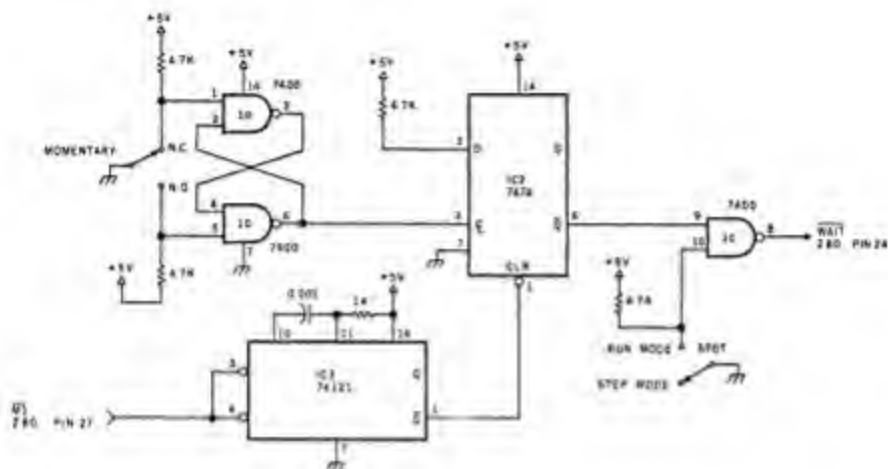


Figure 4.5 An instruction single-stepping circuit.

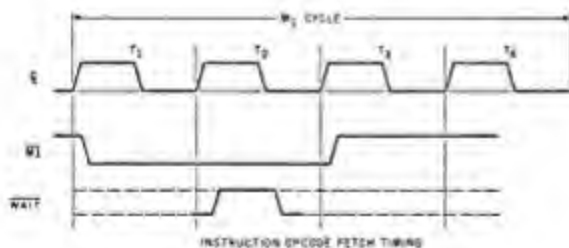
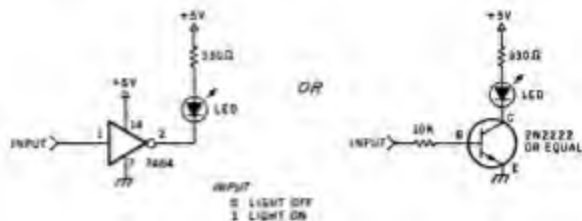


Figure 4.6 Instruction operation-code fetch (M1) timing.

a)



b)

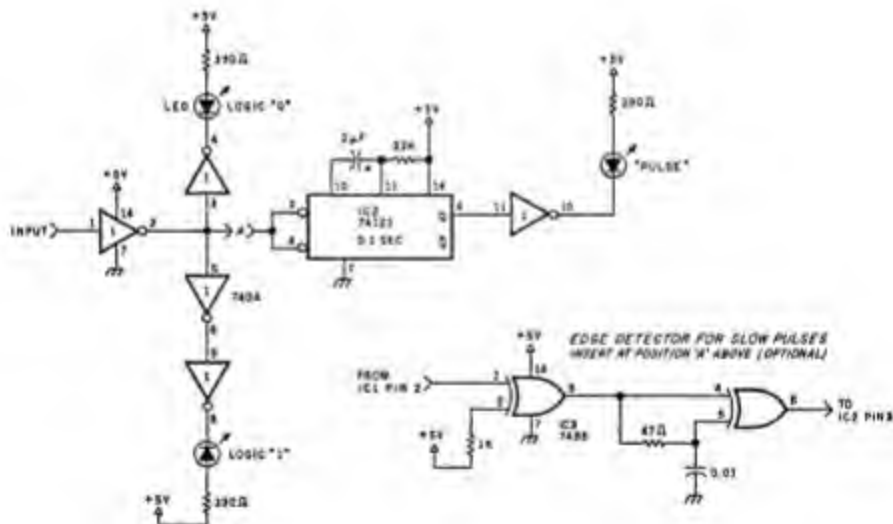


Figure 4.7 Typical LED drivers and a simple logic probe to monitor logic level changes.

- a) Visible logic level indicators that can be attached to the address and data buses to provide a display.
- b) A simple logic probe.

B. Reset Circuit

Often ignored, the reset function is one of the most necessary controls of a computer. Its importance is immediately recognized when running an incorrectly executing program. The reset command on the Z80 stops execution and loads the program counter with 00 hexadecimal (the lowest memory address). This allows the programmer to restart the program. When combined with the instruction single-stepping circuit previously outlined, programs may be started, stopped, and started again at any time.

A reset input can be manual, automatic, or a combination of both. Figure 4.8a is a standard push-to-reset circuit. Its output is normally high until the button is pushed, and then it goes low. The Z80 will remain reset for as long as the button is held and will only begin to execute again when released. Manual reset is a necessity for initial program checkout, and this circuit is employed in the basic ZAP.

When computers are used in applications where no human attendant is present, such as a traffic light controller, the manual reset cannot be used; an automatic reset must be employed instead. Figure 4.8b is the circuit of a totally automatic power-on reset. When power is first applied to the computer, the 10 mF capacitor will be completely discharged. The resultant logic 0 level on the input of the 7404 pin 1 will be maintained for approximately 50 ms, long after the +5 V supply has powered up the rest of the computer. The long charging rate of the capacitor will, in turn, generate a logic 0 (a reset condition) to the computer until the input level rises to approximately 2 V (a TTL logic 1). Once full power is applied, the time it takes the reset circuit to reach 2 V will constitute about a 35 ms power-on *Reset* pulse. Resetting the machine would require turning the power off.

Manual and automatic reset are combined in figure 4.9. This circuit allows the computer to start program execution immediately after power is turned on. The program can be stopped and restarted by pressing the reset button. Slightly different components and additional functions are included in this diagram. Schmitt-triggered inverters (7414s) increase the reliability of the design. When the power is turned off, the use of a diode to discharge the capacitor quickly assures that a pulse will be generated if power is suddenly reapplied. Because power line glitches are usually short in duration, the discharge rate of the capacitor has to be fast enough not to miss generating a reset pulse once power is restored.

While this reset circuit is not necessary for initial computer check-out, it should eventually be employed if ZAP is to be expanded to include any of the options outlined later. To synchronize the central processor and peripherals, they should be tied into the reset signal from this circuit.

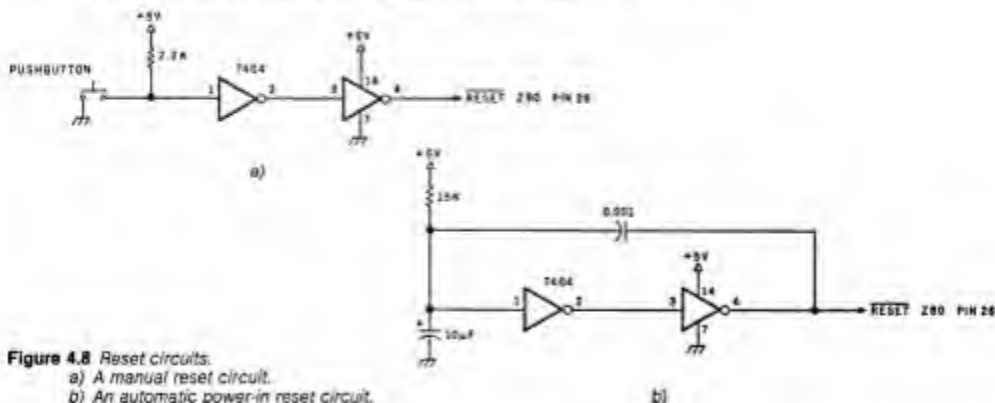


Figure 4.8 Reset circuits.

a) A manual reset circuit.

b) An automatic power-on reset circuit.

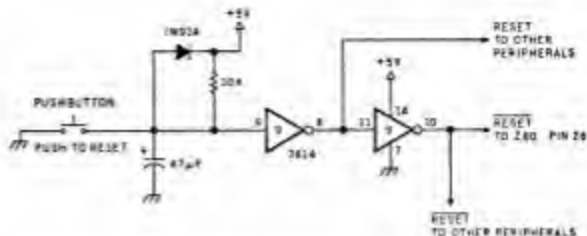


Figure 4.9 A circuit to combine manual and automatic reset functions.

C. Address Bus and Control Output Buffering

The Z80 has the ability to directly address 65,536 (often called 64 K) individual bytes of program memory and 256 individual input and output ports. Because the microprocessor is a binary device, it is only natural that this address be binary. There are 16 binary address lines labeled A0 thru A15. A0 is the LSB (least significant bit), and A15 is the MSB (most significant bit).

The logic levels on this bus are not arbitrary. The control section of the central processor sets the program counter to the next instruction to be executed, and on the fetch cycle, it places the program counter contents on the address bus. During I/O instructions, additional timing cycles place the I/O device address on the 8 least significant bits (A0 thru A7). Because this bus has to drive the inputs of many parallel devices, all of which draw some input power, the address bus must have an output current that will meet the load demand. The Z80 by itself can sink 1.8 mA maximum or one TTL load on each pin. This is no problem if the designer uses low power memories and peripheral interface chips. These are expensive devices, and their use would not necessarily serve to educate the builder in the same way as configurations of less complex circuits.

Using lower density ICs and TTL devices for decoding functions is less expensive but requires considerably more power from the bus. The following table lists the input loading of various devices:

Device	Worst case input current
Standard TTL (7404, 7442, etc)	1.6 mA
Low-power Schottky TTL (74LS04, etc)	0.18 mA
2708 (1K×8 EPROM)	10 μ A
2114 (1K×4 programmable memory)	10 μ A
2716 (2K×8 EPROM)	10 μ A
2102 (1K×1 programmable memory)	10 μ A
8212 (8-bit latch)	0.25 mA
8T97 (6-bit driver)	1.0 mA

It is easy to see that the real power eaters are TTL devices. Low-power Schottky TTL (LSTTL) devices can be substituted throughout the ZAP computer. They save power at slightly additional cost, but the circuit has sufficient power to support straight TTL. If LSTTL is substituted, it must be substituted throughout.

The loading caused by memory, especially with only 2 K bytes in the basic ZAP unit, is insignificant. With 1.8 mA drive current available from the Z80, we could use LSTTL for the I/O and memory address decoding but would have to limit the fanout (total input connections) on each address line to 9 LSTTL inputs. This is sufficient for the basic ZAP and would probably be an

acceptable procedure, but it is not recommended.

The first time a user attaches the logic probe (figure 4.7b) to an unbuffered address line, the computer may die. The load presented by the probe, as well as by the other circuitry, will exceed the drive capability of the bus. It's important that the monitoring devices not impede circuit operation.

Rather than try to optimize the design to a degree that forces the user to be aware of every μA (microampere) consumed by test probes and LED drivers, it's easier to add buffering that increases the bus output power to a point where loading is not an important factor. This is the philosophy behind ZAP busing, and as a side benefit, it will provide enough power to expand ZAP to 64 K should the user ever desire to do so. It also allows the user to add his own TTL circuitry without becoming overly concerned with bus loading.

To achieve high power output from the address bus, a buffering device (called a non-inverting bus driver) is used. The A0 thru A15 outputs of the Z80 make only one connection: to the drivers' input. All other devices that use the address are attached to the output of the drivers.

Figure 4.10 is the diagram and truth table of the 8T97 bus driver. (An equivalent bus driver is the 74367.) This three-state device is capable of sinking 48 mA and can accommodate any combination of TTL, LSTTL, and memory connections a user would want to make. The final address bus configuration is shown in figure 4.11.

The three-state function of the 8T97 is controlled by the $\overline{\text{BUSAK}}$ signal. This signal turns over control of the address bus to an external device during direct memory access operations. In a non-DMA situation, $\overline{\text{BUSAK}}$ is high and the 8T97 passes all outputs from the Z80. When a DMA request is acknowledged, $\overline{\text{BUSAK}}$ goes low, putting the 8T97 in a high impedance output mode. This facility allows memory to be written into or read by an external device and is usually reserved for high-speed operations that are faster than the central processor can achieve.

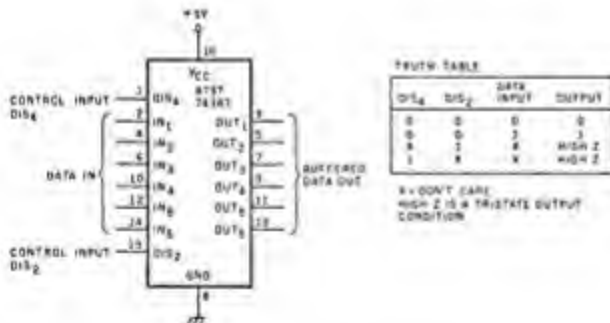


Figure 4.10 The pinout and truth table of an 8T97/74367 bus driver.

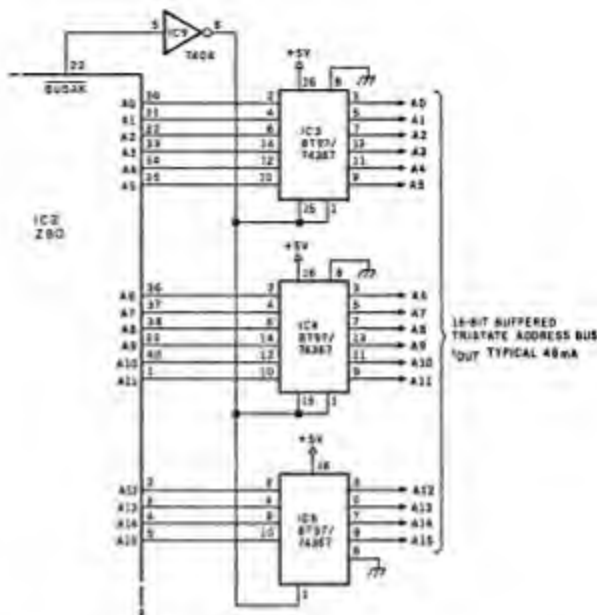


Figure 4.11 The final buffered address bus configuration.

D. Data and Control Bus

The fourth and last area of direct central processor connections is the data bus and the remaining lines of the control bus. The reason for buffering the data bus is similar to the argument for the address bus with one exception—the data bus is bi-directional.

A bi-directional bus means, of course, that data flows in both directions. When the Z80 is writing a byte of data into a memory location, the data flows from the central processor to memory. When the central processor is reading a memory byte, data flows from memory to the central processor. The bi-directional nature of the data bus requires that the bus drivers be either bi-directional internally, or attached in such a way that the same function is performed.

One way of making this bi-directional driver is to use two 8212s. The 8212 (figure 4.12) was originally conceived and produced by Intel as an 8-bit latched input or output port. The 8212 can be latched continuously so that data flows through it, or it can be turned off to block the flow. It is well suited to this application because it has a three-state output.

Two 8212s (figure 4.13) are wired in opposite directions. IC 6 directs data from the central processor toward memory, while IC 7 channels data into the Z80. Control is exercised through a single line connected to the \overline{RD} control signal of the central processor. \overline{RD} is normally low except during write operations. This causes IC 6 to be off, in a three-state mode, and IC 7 on, which allows data from memory or I/O devices to reach the central processor. When \overline{RD} goes high during a write operation, the process is reversed; IC 6 turns on and IC 7 turns off. It is only necessary to use the \overline{RD} line to control data direction. We're assuming, of course, that when the central processor isn't writing data, it must be reading it. While not exactly true, the concept

works well enough in practice, and the two 8212s are connected schematically as in figure 4.14.

It is not absolutely necessary to use 8212s to perform this function. Either 8T97s or 74367s work equally well but take 4 IC packages. If you don't mind the extra wiring and have a source for 8T97s, they can be wired as illustrated in figure 4.15.

The final connections to the central processor to be discussed are the control bus signals, shown in figure 4.16. They coordinate peripherals and channel data and addresses into and out of the central processor at the proper times. Each was briefly explained on the Z80 pinout. Exact timing will be detailed when we discuss attachments of memory, I/O, and enhancements to ZAP. For the time being, unused control inputs are tied high (through resistors) to inhibit false triggering.

The output lines are buffered for the same reasons as was the address bus. Furthermore, because this is a development computer, with expansion in mind, both the inverted and noninverted control signals are brought out to the user.

The areas discussed thus far are combined into a single diagram (figure 4.17) called the Z80 bus and control diagram.

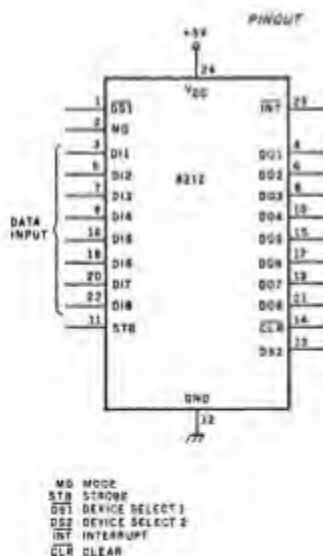
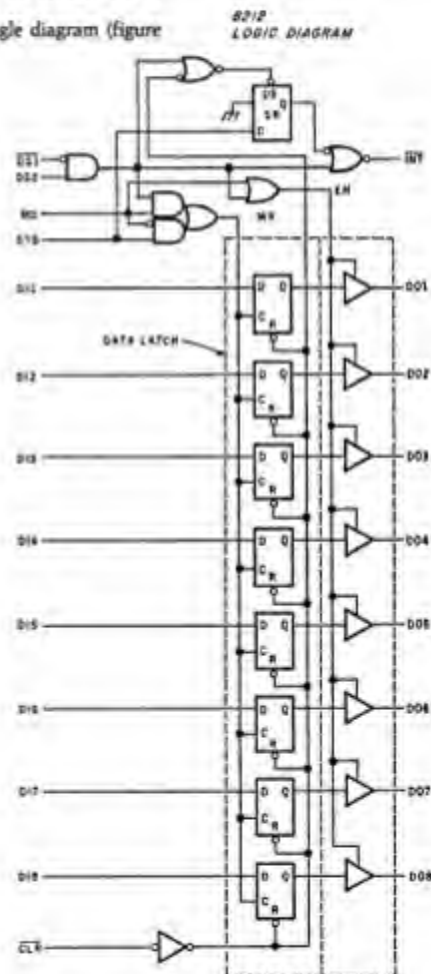


Figure 4.12 The pinout and logic diagram of the 8212 8-bit input/output port.



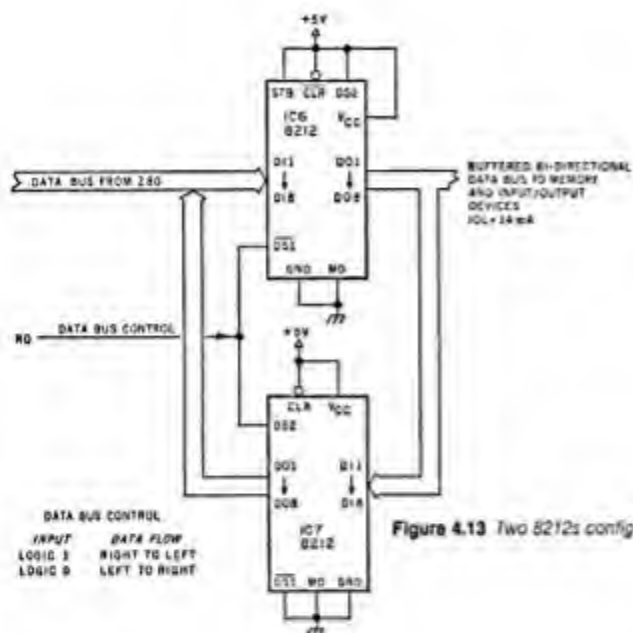


Figure 4.13 Two 8212s configured as bi-directional data bus drivers.

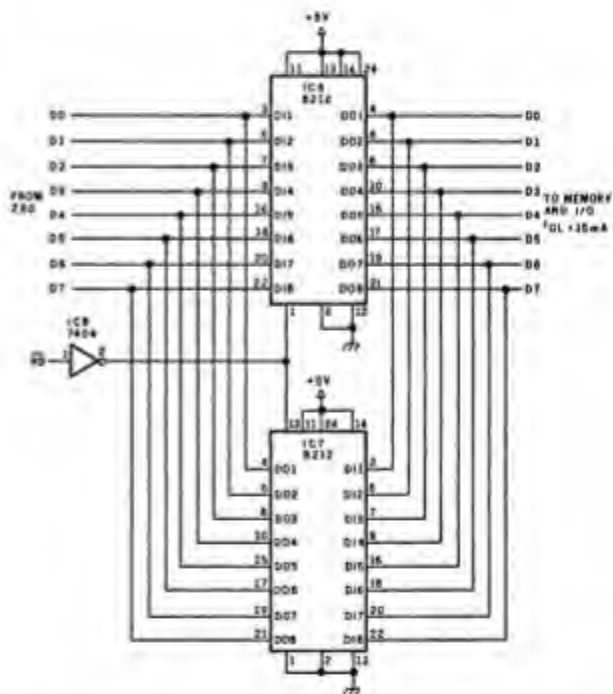


Figure 4.14 A schematic diagram of two 8212 8-bit latches configured as bi-directional data bus drivers.

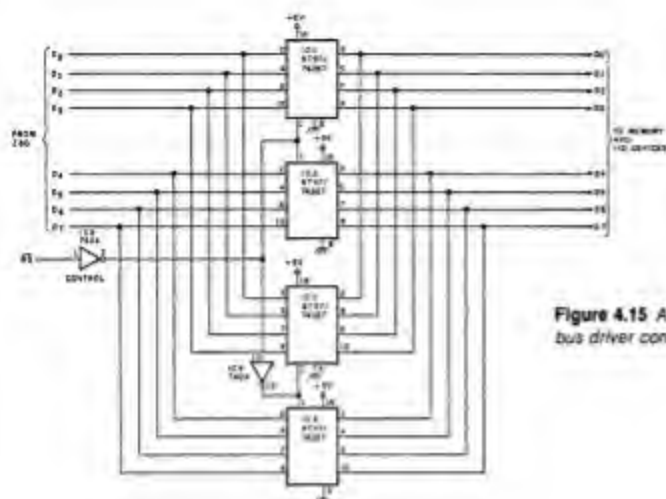


Figure 4.15 A schematic diagram of a data bus driver configured with 8T97s.

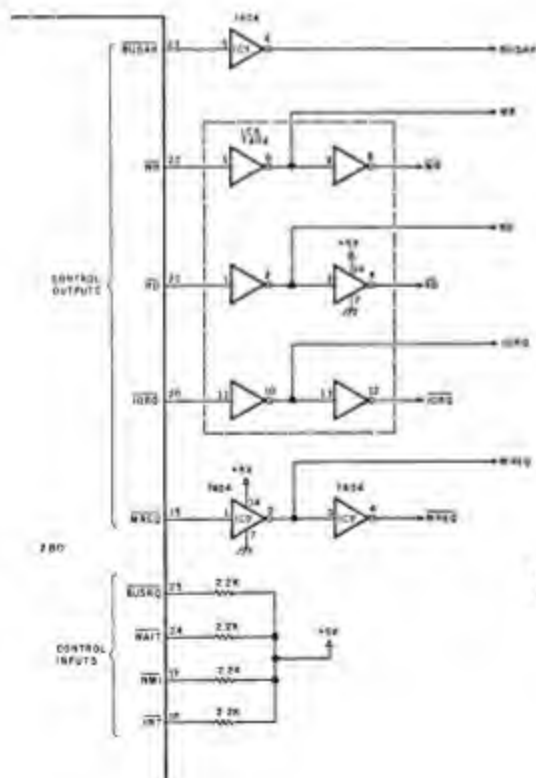
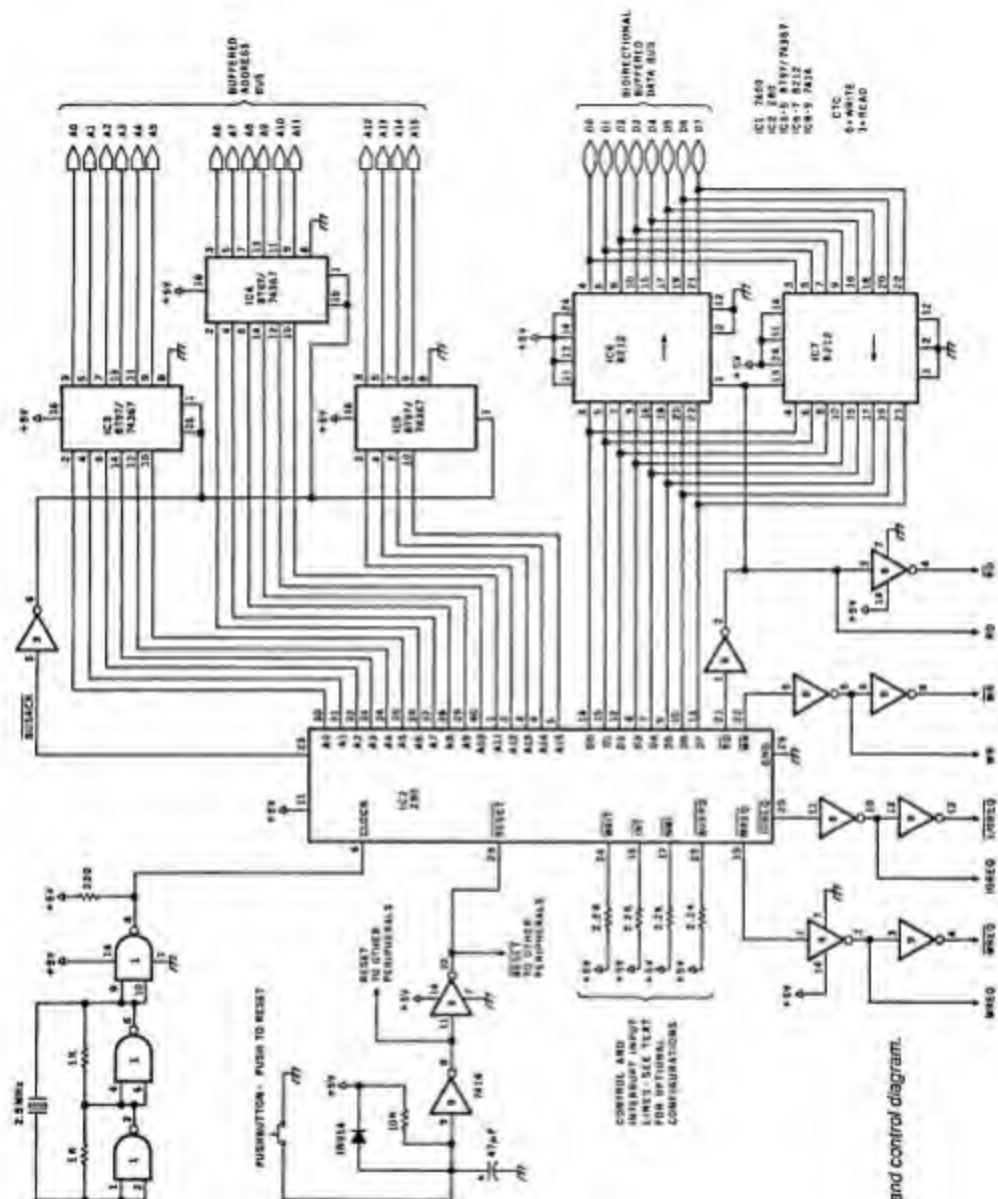


Figure 4.16 Control input connections and output buffering of the basic ZAP design.



E. Testing

Insert all ICs except the Z80 and turn on the power. Each section is then individually tested as follows:

Clock — Testing the 2.5 MHz clock of figure 4.3a will require an oscilloscope or frequency counter to register the exact clock rate. Using the logic probe from figure 4.7b to monitor this clock rate would light all three LEDs. This indicates that the clock functions, but it will not indicate the rate. A similar test can be performed on figure 4.3b.

Single Cycle — The logic probe (without the addition of the 7486 edge detector) is perfect for checking the single-cycle circuit of figure 4.4. With the probe on section C pin 8, the indication should be low. Pressing and holding the button down should change the indication to a high level and cause the "pulse" LED to flash once. Releasing the button should not flash the pulse indicator as it returns to its initial logic condition.

Single Step — With the switch in the single-step mode position (figure 4.5), take a clip lead and momentarily ground IC 3, pin 3. The output at IC 1, pin 8 should be low. Pressing the single-step button will cause this output to go high. It will stay high until IC 3, pin 3 is momentarily grounded again. Check out the pushbutton debouncing circuit (which consists of IC 1 sections a and b) in the same manner as you did the single-cycle test. Finally, with the switch on the run mode, IC 1, pin 8 should always be high.

Power-on Reset — The circuits of figures 4.8a and 4.8b should have a normally high output. When power is first applied to figure 4.8b, or the button pressed in figure 4.8a, the output should go low. Either situation will cause a logic low level to occur from the circuit of figure 4.9.

Address Bus Drivers — *The Z80 should not be inserted!* With IC 9, pin 5 grounded, all outputs of ICs 3, 4, and 5 on schematic figure 4.11 should appear high. In actuality, this will be the three-state output mode and the proper test equipment will register them as open circuits. Tying IC 9, pin 5 to +5 V through a 2.2 K resistor will turn on all the bus drivers. Their outputs will all be logic high levels. Successively grounding the A0 thru A15 lines at the Z80 connector should result in a low-level indication on the respective buffered output line. When all 16 lines can do this successfully, the address bus checks out.

Bi-directional Data Bus — The data bus is tested in a similar manner except that the procedure is done twice—for data flow in either direction. Grounding IC 8, pin 1 (figure 4.14) simulates a read condition. Data should flow from right to left. Applying ground and +5 V (through a 2.2 K resistor) alternately to the data input pins of IC 6 should produce similar levels on DO1 thru DO8 of IC 6. Raising IC 8, pin 1 to +5 V allows similar data transfer, but only from left to right this time.

Control Bus — Referring to the schematic of figure 4.16, testing is simply a case of applying a known logic level to the input side of the series inverters and noting the output levels one gate at a time. For example, if Z80 pin 19 was a logic low, IC 9, pin 2 would be a logic high and conversely, IC 9, pin 4 would be low. Each inverter section which the signal passes through inverts the signal.

II. Memory and I/O Decoding

Before we can utilize the memory or I/O devices we must learn how the Z80 addressing works. Remember, the address FF hexadecimal could refer to memory, or an input or an output port. The computer must have the ability to differentiate among the three

possible meanings.

The control outputs of the Z80 contain the necessary routing information, and by properly gating them together, the correct signals are obtained. For basic I/O and memory operations, the four signals of particular interest are $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$. Their definitions are as follows:

- A. $\overline{\text{MREQ}}$
Memory Request. Whenever a transaction occurs between the central processor and memory, the $\overline{\text{MREQ}}$ line goes to a logic 0.
- B. $\overline{\text{IORQ}}$
Input/Output Request. Whenever a transaction occurs between the central processor and either an input port or an output port, the $\overline{\text{IORQ}}$ line goes to a logic 0.
- C. $\overline{\text{RD}}$
Read Request. Whenever the central processor reads input data from either memory or an input port, the $\overline{\text{RD}}$ line goes to a logic 0.
- D. $\overline{\text{WR}}$
Write Request. Whenever the central processor is writing data to either memory or to an output port, the $\overline{\text{WR}}$ line goes to a logic 0.

To differentiate between input and output ports during I/O instructions, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ are gated together as shown in figure 4.18. In a similar manner, $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are gated during memory transfers as shown in figure 4.19. Unlike the I/O decoding, but similar to the address bus driver discussed earlier, a memory-read condition does not have to be decoded. It is assumed that when the memory is not in a write mode, it is in the read state.

The resulting three decoded strobes define the operations of Input Port Read ($\overline{\text{IPRD}}$), Output Port Write ($\overline{\text{IOWR}}$), and Memory Write ($\overline{\text{MEMWR}}$). If only three functions were required in your particular computer configuration, then no other decoding would be necessary. Such a computer would have one input port, one output port, and one bank of memory. To alleviate this problem, additional decoding of I/O and memory is necessary so that these control strobes can serve more than a single device. With the extra circuitry, the Z80 can independently address 256 input and output ports and 64 K bytes of memory.

During an I/O request (either input or output), the 8-bit binary address of the particular I/O port appears on lines A0 thru A7 of the address bus. An explanation of address coding is shown in figure 4.20. Additional examples are illustrated in figure 4.21.

Using this information, if an instruction were to designate output port 7 as its destination, then the circuitry of figure 4.22 could be used. When a code of 007 octal (07 hexadecimal or 00000111 binary) appears on the address lines with an $\overline{\text{IOWR}}$ strobe, the signals present on the data bus would be stored in an 8-bit register as output data.

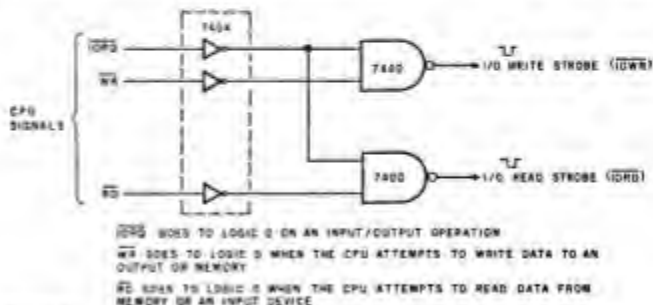


Figure 4.18 Input/output read and write decoding.

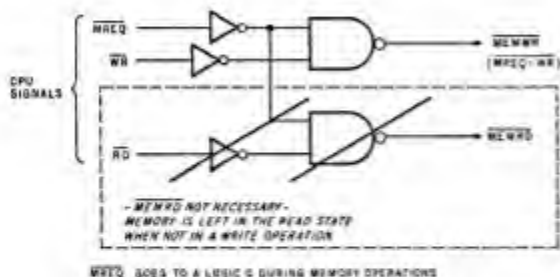


Figure 4.19 Memory read and write decoding.

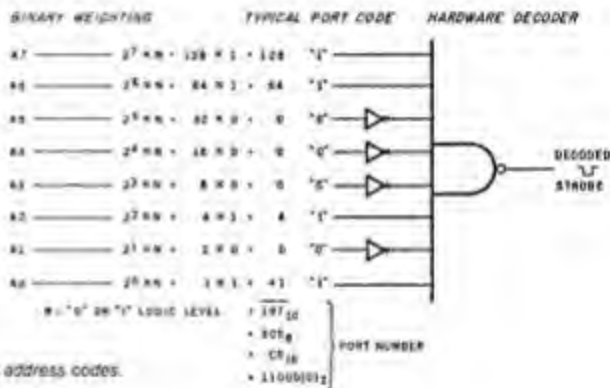


Figure 4.20 An explanation of input/output address codes.

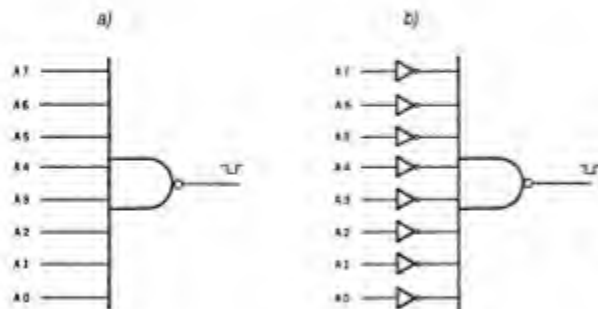


Figure 4.21 Address decoding logic.
a) For address FF₁₆.
b) For address 00₁₆.

addresses. This is not a problem as long as the user is aware of repetitive addressing and watches his programming. Should more than 8 stobes be required, the 7442 can be replaced with a 74154 (4 to 16 decoders). This will give 16 I/O port stobes that repeat every 16 addresses.

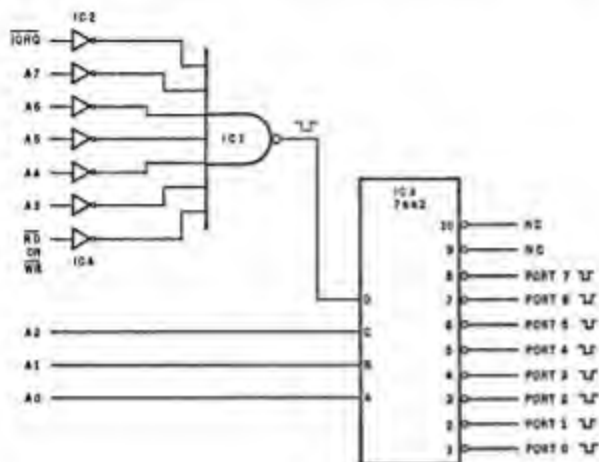


Figure 4.23 A formal input/output port address decoding method that decodes all 8 address lines.

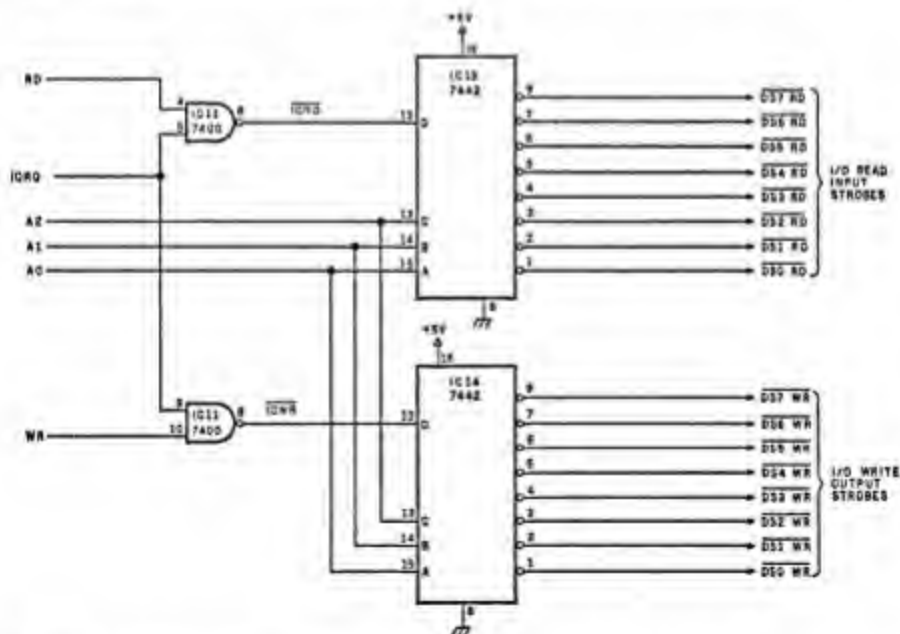


Figure 4.24 A method for decoding input/output stobes with a reduced amount of circuitry.

Memory Decoding

Decoding the memory address bus is accomplished in a similar manner. It is inadvisable to take the same tack and allow repetitive memory addressing because there is more likelihood of error. Even though 16 lines are involved, in actual application, memory decoding turns out to be less complicated. ZAP uses $1\text{ K} \times 8$ -bit banks of programmable memory and 1 K-byte erasable read-only memory. Both of these devices require 10 address lines to define the 1 of 1024 locations in each bank. This leaves only 6 lines that have to be individually decoded to define any 1 K block of memory. Figure 4.25 illustrates how this can be accomplished. A 7442 (4- to 10-line decoder) is used to generate 8 separate chip-select lines. Because the address lines of the 7442 are tied to A10 thru A12, each strobe pulse will have a boundary of 1 K. It is not by chance that $1\text{ K} \times 8$ was chosen as the memory capacity of each bank.

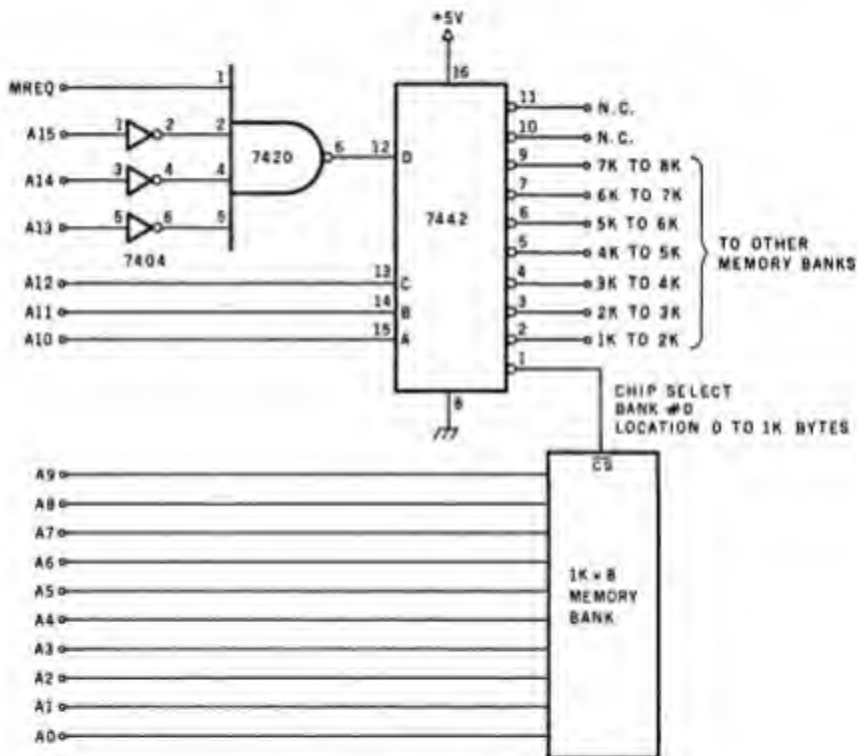


Figure 4.25 Memory bank decoding for 8 K of memory.

While the basic configuration of ZAP provides decoding for 8 K of memory and 8 input and output ports, not all of these chip selects and port strobes are used. The extra lines are left for expansion. Figure 4.26 is a completed schematic of the I/O and memory decoder for the builder to add to the circuit in figure 4.17.

Testing

After you have added the components of figure 4.26 to figure 4.17, you are ready to test the memory and I/O decoding. Insert ICs 10, 11, 12, 13, and 14, but don't insert IC 20 yet. ICs 1, 3, and 9 should remain inserted from the previous test. The Z80 should still be left out. The logic level at the D address input of each of the 7442s (ICs 12, 13, and 14) should be high. Pulling out ICs 8 and 9 (with power off) will cause this input to immediately change to a logic low level.

Next, ground pins 30, 31, and 32 and tie 23 high on the Z80 socket. With the address bus buffers enabled, and a 000 address jumpered on A0 thru A2, a chip-select low should appear on the lowest strobe address. In this case, pin 1 of ICs 13 and 14 should be low and the other strobe lines high. Changing the 3 jumpers on A0 thru A2 will enable other device chip-select strobes. The memory bank decoder works the same way except that the jumpering should be applied to address lines A10 thru A12.

After testing, insert all chips except the Z80.

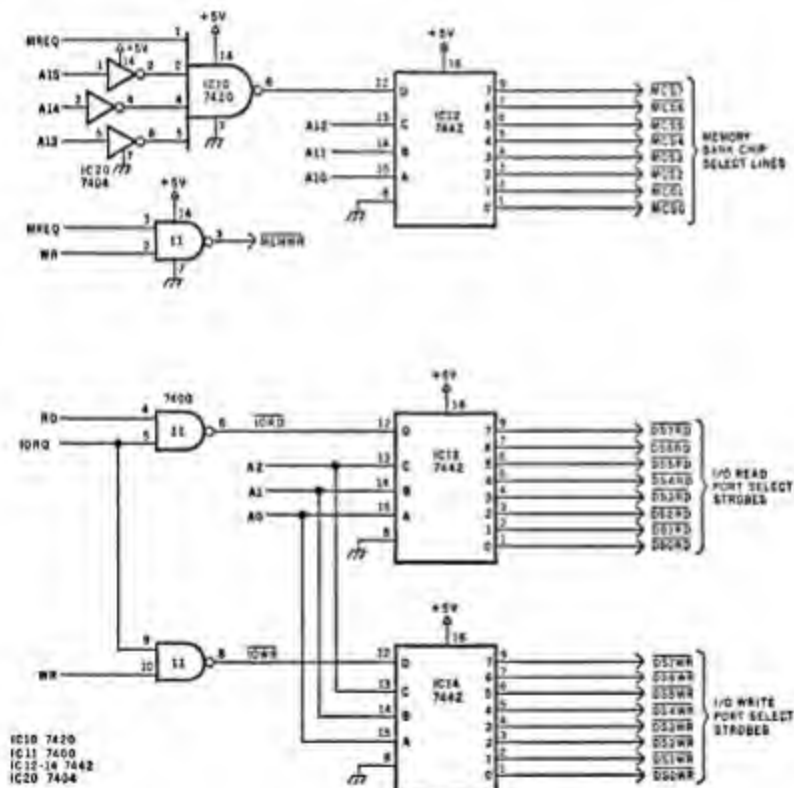


Figure 4.26 The memory and input/output decoding section of ZAP.
a) Memory bank chip-select strobes.
b) Input/output device chip-select strobes.

III. Memory

Of course, a major consideration for any computer system is memory. Both program instructions and data must be stored and recalled at the appropriate time so the computer can perform its function. Even though the Z80 central processor has a quantity of 8-bit storage registers, these can be only used for temporary manipulation of data and cannot store program instructions. Program instructions must be stored in external memory elements.

The external memory may be divided into two broad classes: ROM (read-only memory) and RWM (read/write memory). ROM is used to store specific, unchanging program steps or data. The contents of these memory locations are considered permanent and cannot be easily changed. Read/write memory, on the other hand, is used to store data that changes while the computer is operating. Examples would be the results of calculations or programs that change frequently. For either type of memory, the ultimate function is still the same; to provide, on demand, either an instruction for execution or a location where data may be stored.

Read-Only Memory

ROM (read-only memory) is an important part of the computer system. ROM functions as a memory array whose contents, once set by special programming techniques, cannot be altered by the central processor. There are few exceptions to this rule.

By its nature, ROM is non-volatile. When power is turned off, the program contents are not lost. Reapplication of power allows immediate program execution.

Within this basic category of ROMs there are three subcategories — ROM, PROM, and EPROM — which are defined more by usage and application than their names might imply.

ROM — Read-Only Memory

This is storage which can be written into only once. The information is fixed and cannot be changed. A ROM is usually mask programmed by the manufacturer and is bought with a preset bit pattern. These types of ROMs are considered to be custom programmed.

PROM — (User) Programmable Read-Only Memory

This storage can also be written into only once and the information is fixed. These devices are typically bipolar fusible link PROMs, which are programmed by the user rather than the manufacturer. ROMs and PROMs do not generally use the same semiconductor construction technology. Storage is much denser on a ROM than on a PROM, and cost-per-bit is generally lower on a ROM.

EPROM — Erasable-Programmable Read-Only Memory

This device combines the best parts of a ROM and a PROM. When received from a manufacturer, all storage locations are unprogrammed. Using a special interface, the EPROM can be programmed by the user as a PROM would be, with the result utilized as a ROM. If the EPROM content must be changed, it can be erased and reprogrammed. Depending upon the particular device, an EPROM can be either electronically alterable (often differentiated by the separate abbreviation EAROM) or ultraviolet erasable. The latter is sometimes called a UVEPROM, but is more often just called an EPROM. They are easily recognizable because they have a quartz window over the integrated circuit. This window is transparent to ultraviolet light and facilitates erasure.

While there can be considerable discussion as to the merits of each option, all ROMs perform the same ultimate function. For each independently addressable location, there is specific stored-bit pattern. Only the processor can determine whether this is data or an instruction. The method of storage is the same in either case. Figure 4.27 details the block diagram of a ROM.

A ROM is simply a logical block which, under program control, provides a preset

pattern. Figure 4.28 is a 3-bit read-only memory. When switch SW1 is closed (the position it would take when the central processor wanted the stored information), the 3-bit code of "101" would appear at the outputs. The diode grounds the input signals to the 7404 inverters when SW1 is closed. Expanding to more than 3 bits is simply a matter of adding more diodes, resistors and buffer stages. Such a circuit is referred to as a diode-matrix ROM and in this case would be a 1-line by n-bit ROM.

A 3-bit memory is not much use. This concept can easily be expanded to 16 bytes by adding an address decoder as diagrammed in figure 4.29. A completed schematic with the diodes specifically arranged to perform a simple 9-byte program is illustrated in figure 4.30. This short test program will be used later during the checkout phase.

The diode-matrix ROM is presented for its educational value only. This is not a method that should be employed in the ZAP computer. Realizing that there are integrated circuits that would successfully fulfill the requirements in each of three categories, we must analyze our needs a little more closely.

The pertinent questions are: memory size, and the cost and ease of programming. The size of a ROM is determined by the user. When power is first applied, how much effort does the user want to expend to make the computer execute a specific program? ZAP has no front panel and no banks of address and data switches to toggle in instructions. This being the case, ZAP must have a program that executes immediately (when power is applied or the reset button is pushed), and that allows the central processor to communicate with its peripherals and set itself in a mode that is directly programmable through these devices. Once power is applied, a simple 50- to 100-byte program can be written, which facilitates keyboard to memory loading. But perhaps we need to enter a large program in memory? Are we to enter it all through the keyboard?

High-speed data entry can be accommodated through a serial interface. This can be added at the expense of another 100 or 200 bytes. Another consideration is the necessity for some operator address and data display to ease program development.

In conclusion, to incorporate all the functions necessary for a single-board development system, the ROM can easily require 500 to 1,000 bytes of storage. Many computer systems use a 64- to 256-byte ROM to store a bootstrap program. A bootstrap is a program that coordinates the minimum amount of necessary peripherals to load a larger program into the computer. In most personal computer systems, this bootstrap controls a cassette interface, and the program that is subsequently loaded is called a monitor.

A monitor (explained in Chapter 6) is a very important piece of software that requires about 1 K of program storage. Our decision is whether to make the monitor totally resident in ROM (ready for immediate execution), or to reduce ROM to the barest minimum and load the monitor from either a keyboard or a cassette storage system.

This is an important consideration for someone building a computer from scratch. When given a choice, I feel, you should almost always opt for the solution that calls for the fewest components and you should include the ROM monitor in the hardware. It's like putting the cart before the horse to require that a cassette interface be used to load all the diagnostic software. It's quite possible that the monitor program, resident in a 1 K ROM, would be required to troubleshoot and align the serial interface and cassette modem sections. A further consideration is that the ZAP computer can be brought on line sooner. With a ROM monitor, useful programs can be entered via the keyboard without having to build a serial interface.

I suggest that the preferred ROM memory size for ZAP be 1 K. As previously mentioned, ROM is mask-programmed by the manufacturer. However, let's not forget that for a home-built computer, you are the manufacturer. Fusible link PROMs are an expensive proposition when configured in a 1 K block. As a 64-byte bootstrap loader they are ideal.

The suggested alternative for the ZAP read-only memory is to use an EPROM that is programmed by the user. A 1 K EPROM such as the 2708 (or the 2 K 2716) is cost-effective for the home-built computer. The Intel 2708 ultraviolet erasable read-only memory is recommended for this application. (The 2716 is a 2 K EPROM with a single +5 V power supply.)



Figure 4.27 A block diagram of a read-only memory.

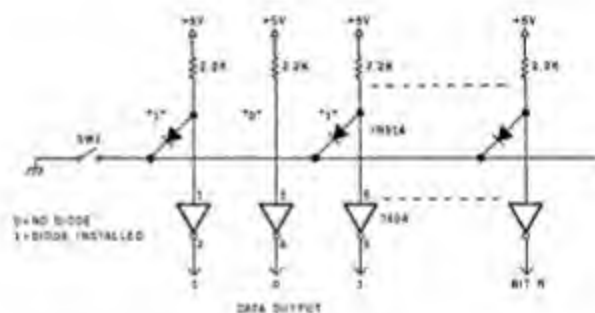


Figure 4.28 A simple 3-bit read-only memory (1 x 3 bits).

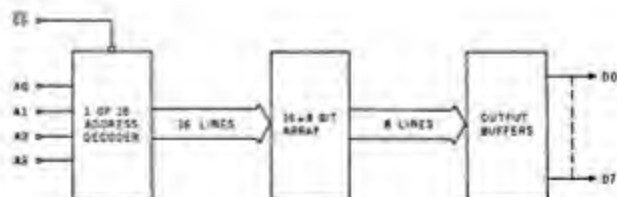


Figure 4.29 A block diagram of a 16-byte read-only memory.

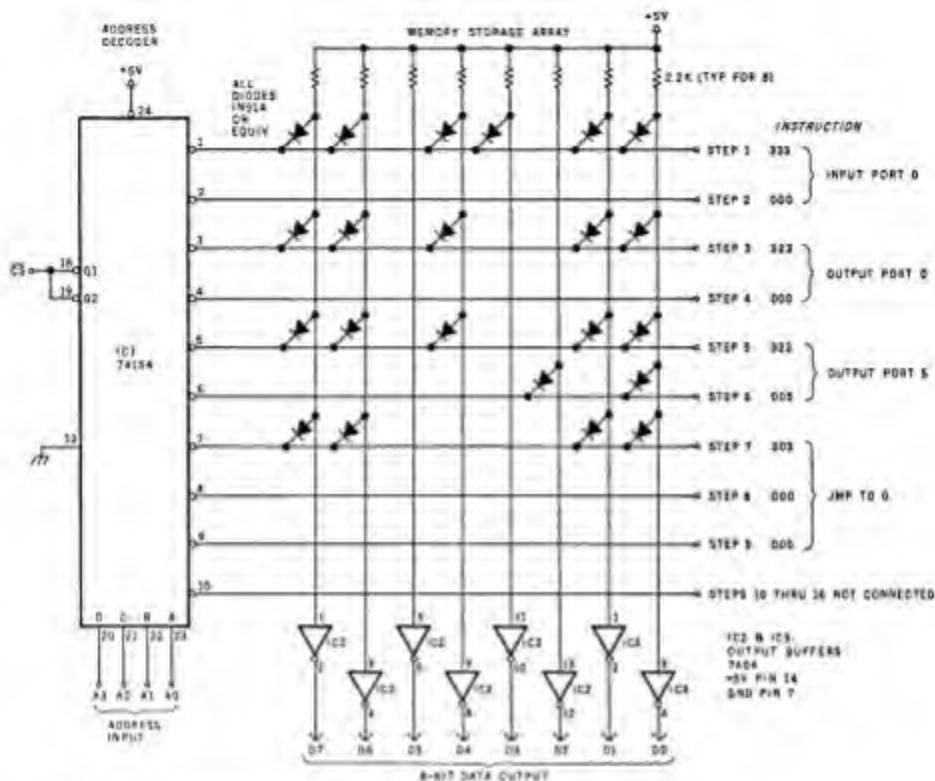


Figure 4.30 A diode-matrix read-only memory with a test program

EPROMs

The EPROM is a read-mostly memory. It is used as a ROM for extended periods of time, erased occasionally and reprogrammed as necessary. Erasure is accomplished by exposing the chip substrate, covered by a transparent quartz window, to ultraviolet light. The EPROM memory element used by Intel in the 2708 is a stored-charge type called a FAMOS transistor (Floating-gate Avalanche Injection Metal Oxide Semiconductor storage device). It is similar to a p-channel silicon gate field-effect transistor with the lower or "floating" gate totally surrounded by an insulator of silicon dioxide. The 1 or 0 storage value of the FAMOS cell is a function of the charge on the floating gate. A charged cell will have the opposite storage output of an uncharged cell. By applying a 25 V charging voltage to selectively addressed cells, particular bit patterns that constitute the program can be written into the EPROM. Surrounded by insulating material, the charge can last for years. When this silicon dioxide insulator is exposed to intense ultraviolet light it becomes somewhat conductive and bleeds off the charge on the floating gate. The result is erasure of all programmed information.

Appendices C1 and C2 detail the pin layout and electrical specifications of the 2708 and the 2716 respectively. Chapter 7 explores various methods to program and test the chip.

Read/write memory is just what its name implies. Such memory allows data to be written into it as well as be read from it. Read/write memory for microcomputers is generally configured from semiconductor programmable memory devices that retain data only while the power is on.

ROMs are technically random access devices; however, read/write memory, which is composed of semi-conductor devices and is primarily intended for use in microcomputers, has come to be called RAM (random access memory). From this point on, we shall refer to RAM as programmable memory.

There are two classes of programmable memories: static and dynamic. Static programmable memory stores each bit of information in a bi-stable storage cell such as a flip-flop. This information is retained as long as the power is supplied to the circuit. Dynamic programmable memories have a simpler internal structure, smaller size, dissipate less power, and are inherently faster. They store information as an electric charge on the gate to substrate of a MOS transistor. This charge lasts only a few milliseconds and must be refreshed. This necessity to refresh the stored information is one of the major distinctions between static and dynamic programmable memories.

Refreshing dynamic memories can be bothersome, however. The process requires that all storage cells be addressed at least once every few (usually 2) milliseconds. A counter circuit is usually incorporated to exercise the memory address lines when the computer is not accessing memory. In most systems, memory refresh requires additional external circuitry. The Z80 contains this circuitry within the central processor chip and greatly facilitates the use of dynamic memory. However, this facility is lost when the Z80 is reset. Therefore, extra refresh circuitry is necessary.

The choice between dynamic and static programmable memory technology is predicated on cost and convenience. Even with the expense of external refresh circuitry, dynamic memory is less costly. In a prototype system such as ZAP, however, dynamic memory is more trouble than it is worth. Once built and operational, dynamic memory might well be the best answer to memory expansion. But at this point in the building process, the inclusion of dynamic memory would over-complicate the design. This book, which emphasizes getting a beginner on-line, deals exclusively with semiconductor static programmable memory applications.

Static Programmable Memory

Figure 4.31 is a block diagram of a static programmable memory element typical of the type used in the ZAP computer. There are five basic components of a programmable memory: 1) address input lines, 2) data input, 3) data output, 4) chip select, and 5) a read/write- or write-enable strobe line. The address input lines are connected to the address bus of the computer. In the case of a N by M bit programmable memory, where N is the number of words and M is the length of each word, there must be enough address lines to address all N bytes. For example, in a 1 K programmable memory it would take 10 bits to address all 1024 bytes within this memory (eg: $2^{10}=1024$). Static programmable memory chips that contain fewer bytes of data, such as a 64-byte programmable memory, would obviously require fewer address lines. For a 64-byte memory, only 6 bits of address are necessary.

Because the function of a static programmable memory device is to allow storage and retrieval of data, provisions must be made for data input and data output from the device. The data input and data output lines (shown in figure 4.31) are designated as separate functions.

During the read function, the stored data within the addressed memory cell is available on the data output lines. During the write function, data that is placed upon the data input lines would be stored at the address designated by the code on the address input lines. It is not necessary that static programmable memory devices have independent data input and data output lines.

In most cases, these devices are configured with three-state outputs. Data input and data output can be attached together to a bi-directional data bus, or they can be the

same lines and time multiplexed. Figure 4.31 illustrates a three-state method of data busing. During a read function, the data input lines are disabled internally within the memory device. The contents of the memory cell addressed by the address input lines are available on data out and are fed directly to the bi-directional data bus. During a write function, the opposite is true. The data output lines are set in the three-state mode (which you may recall is effectively an open circuit), and draw no current from the bi-directional data bus. The contents of the bi-directional data bus are stored at the designated memory cell.

All of these multiplexing functions are dependent upon the read/write and chip-select lines. No operation can occur without the memory device being selected through the chip-select line. To select a particular bank, as outlined earlier, it is necessary to have decoding logic that enables these banks through the chip-select lines. Once a chip or bank of chips has been selected, the computer determines whether data should be read from or written into these memory locations. Under normal operation all static programmable memory is left in the read state, and only enabled during a write command by setting a level 0 on the write enable. This is called a write-enable strobe.

Figure 4.32 is a detailed timing diagram of the memory read and write cycles. The write/enable is a combination of memory request and write. A read/enable is a combination of memory request and read. Proper decoding of these signals and the chip select were discussed previously. In its basic form, ZAP has 8 chip-select lines, each addressing a 1 K bank of memory.

Figure 4.33 illustrates the memory map of the basic ZAP computer. As initially configured, ZAP contains 3 K bytes of memory. Location 0 thru 3FF is a 1 K EPROM. Locations 400 thru BFF are static programmable memory locations. The 1 K EPROM is configured to reside in locations 0 thru 3FF so that ZAP can be easily started with a power-on reset. Programmable memory located at locations 400 and above is considered to be user programmable memory. At least 2 K is recommended for satisfactory operation. ZAP will work with 1 K, but 2 K is recommended for basic peripheral expansion.

Figure 4.33 also shows how memory is attached to the computer. All three banks of memory are attached in parallel between the address and data buses. Each bank has a separate decoded chip-select. When the EPROM is enabled and $\overline{MCS0}$ is at a logic level 0, EPROM data is impressed upon the data bus lines. The other two banks of memory are in the three-state mode and have no effect on the bus. When the computer accesses programmable memory, the chip for that particular bank of memory is set to a logic 0, and only that bank of memory has access to the data bus.

While all banks of memory would have the same address applied to them, only the selected bank would be in the active mode. The logic flow is similar for the computer to write into a bank of memory. You will notice that there are write-enable lines leading to each of the 1 K static programmable memory banks, but not to the 1 K EPROM. A 1 K EPROM can only be written into with a special interface. Therefore, the write-enable strobe is only attached to the programmable memories.

If, for example, the computer were to write into location 400, the chip-select for bank 1 and the write enable for bank 1 would both have to be at a logic 0 to allow data on the data bus to be stored into location 400. This type of programmable memory configuration is both multiplexed and three-state. In the read mode, data flows from the programmable memory chip; in the write mode it flows into it, and when not selected it's three-state.

Up to this point, we have discussed block diagrams of static programmable memory. To produce an operational computer, it's necessary to configure this memory with actual parts. Unfortunately, single chip 1 K by 8-bit programmable memories were extremely expensive when ZAP was designed. Therefore, these 1 K blocks are designed from multiple components. Two relatively inexpensive and popular static programmable memory chips are the Intel 2102A (Appendix C3) and the Intel 2114 programmable memory (Appendix C4).

The 2102A is a 1 K \times 1 static programmable memory. Configuring a 1 K \times 8 block of memory requires eight 2102s attached in parallel. By comparison, configuring a 1 K \times 8 block with 2114s would require only two chips. This is because the 2114 has a higher internal density than the 2102. Because the objective of any hand-wired comput-

er project is to get the device on line easily, 2114s are the recommended programmable memory devices for ZAP. While 2102s will work, the added wiring necessary to use these devices far outweighs the additional cost of the 2114s.

Figure 4.34 illustrates how two 2114s are attached together to produce a $1\text{ K} \times 8$ programmable memory bank. They share a common chip-select line. The data input lines are divided so that 4 bits of data are stored on each chip. Because each has a 1024-byte address capability, the 10-bit address lines are commonly shared. To build the basic ZAP, two circuits of the type illustrated in figure 4.34 should be constructed. The total memory for the basic computer is 3 K. It can be expanded to 8 K without additional address decoding. It is not absolutely necessary to have 2 K of programmable memory if the user wishes only to check the operation of the system. At a minimum, the EPROM must be wired as 1 bank of memory.

The 1 K EPROM contains the monitor which allows ZAP to function. This monitor contains many smaller programs that are called subroutines. When the main program calls a subroutine, it places the return address on a software stack located in programmable memory. At the conclusion of the subroutine, the central processor pulls this address from the stack and returns to the main program. Usually the stack requires no more than 64 bytes. However, it is no less trouble to wire two 2114s for a full $1\text{ K} \times 8$ bank of memory than to try to wire a 64-byte memory.

An additional bank of 1 K, designated as bank 2, could be added at the user's discretion. This bank is necessary if you plan to write programs that will occupy more than 1 K of memory including the stack. As the computer is presently configured, 1 K may appear adequate; however, for the additional programs outlined in this book, 2 K is recommended. This is especially true when a buffer area is required to communicate with external peripherals. The schematic for the final memory configuration is shown in figure 4.35. It should be added to the circuitry of figures 4.17 and 4.26.

Unlike the other sections of the computer, the memory cannot be checked except under program control. Theoretically, the address lines can be preset and data read or stored, but it's not worth the effort. Memory checks will occur after the input/output section is wired. Basically, it will be checked first with EPROM alone, then with the addition of the programmable memory. I mentioned previously that EPROM and programmable memory are related yet operate independently. While a program is often stored in FROM, it usually requires programmable memory for proper execution.

In a short program that loads the accumulator, writes to an output port, and jumps back to itself again, with no subroutine calls, programmable memory is not necessary. It can be completely located on EPROM. The exact procedure for this test will be outlined at the end of the I/O section.

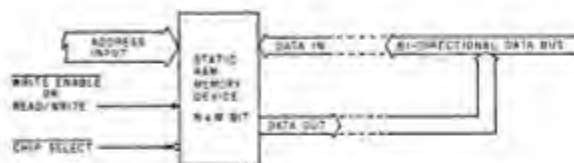


Figure 4.31 A block diagram of a static programmable memory element of $N \times M$ bits.

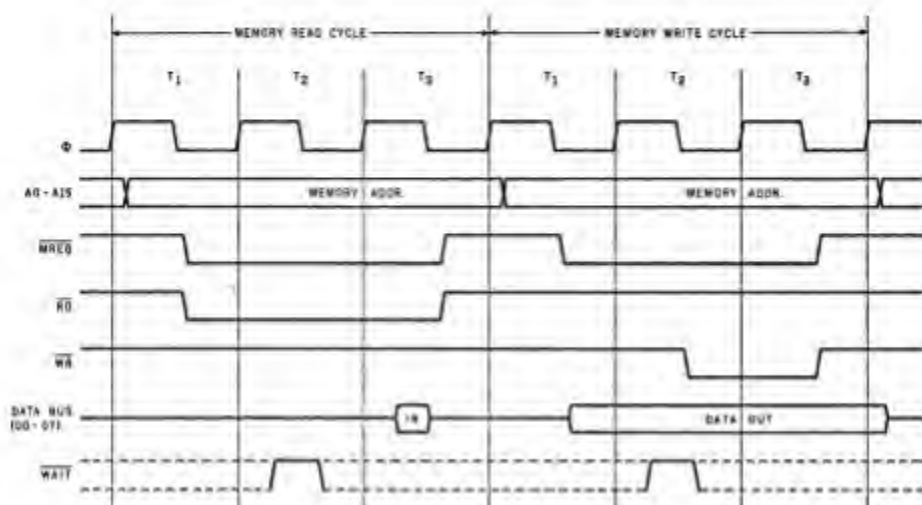


Figure 4.32 A timing diagram of the memory read or write cycles for the Z80. This diagram does not include WAIT states.

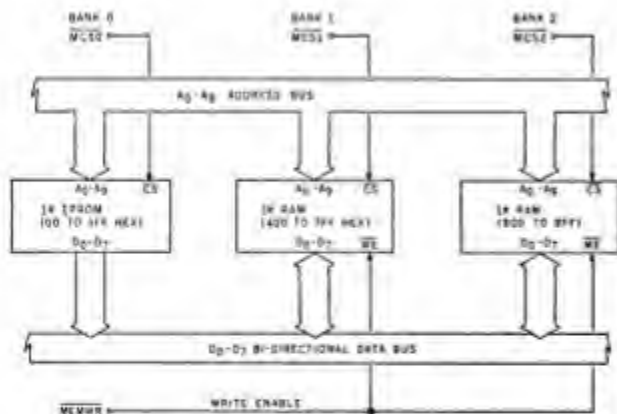


Figure 4.33 A block diagram of the memory map for the ZAP computer.

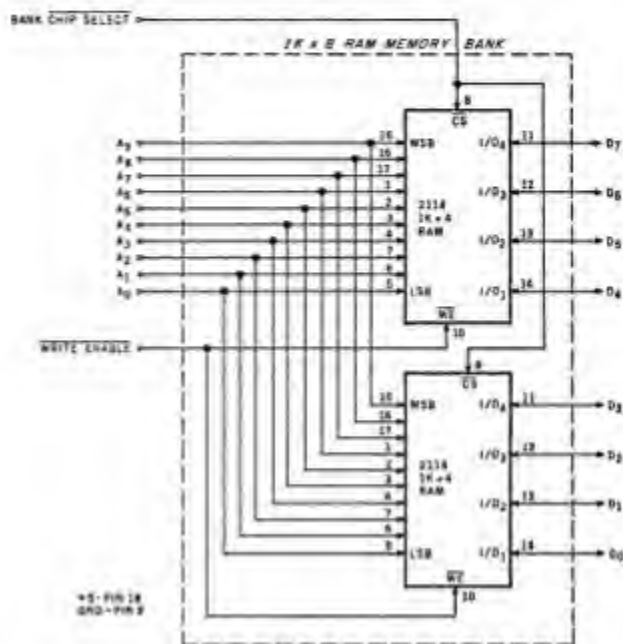


Figure 4.34 A 1K x 8 programmable memory bank constructed by using two 2114 1K x 4-bit programmable memory chips.

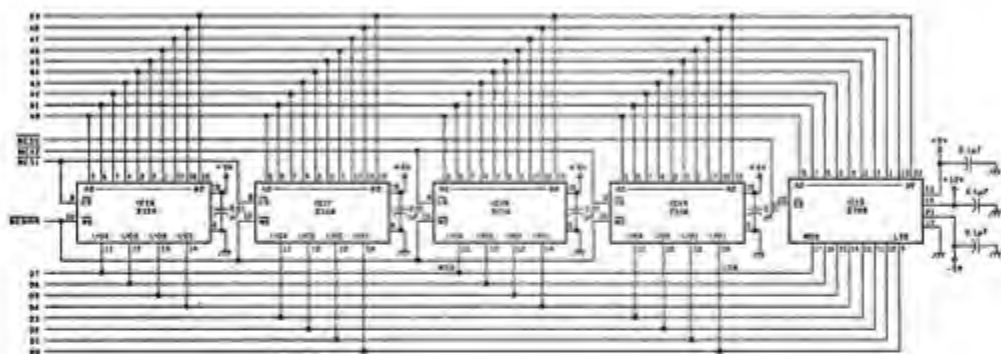


Figure 4.35 A schematic diagram of the final memory configuration for the basic ZAP computer.

IV. Input/Output

Thus far we have discussed the central processor control and memory decoding. The input and output functions are equally important. For the computer to display useful information, it must be "interfaced" to peripherals. "Interface" is an overworked term that refers to a capability of communicating with external devices such as keyboards, video or LED displays, and memory storage systems. Communication can be either data input or output.

Input data can come from keyboards, audio cassette mass storage, or special data acquisition interfaces. Similarly, output data flows from the computer to peripherals (eg: video displays, numeric readouts, printers, and external control interfaces). The function and format of the data communication between the central processor and the peripherals might vary considerably, but the internal routing of the data is fundamentally the same.

The Z80 microprocessor provides both an input and output instruction. An output from the processor is logically the same as writing to memory, and receiving an input from an external device is similar to a memory-read command. They are differentiated from memory operations by gating the read and write status lines with the I/O request control line. Logical concurrence of an I/O request and a read or write status output designates the direction of the communication with the peripheral device. Simultaneously with the control signals, the address code (1 of 256) of the subject device is placed on the address bus. A timing diagram of these signals is shown in figure 4.36. The decoding logic was detailed in section II of this chapter.

Wiring the I/O ports for ZAP is a two-stage process. When hand wiring a computer, the most important consideration is to see that the input/output function works by the least complicated method. A successful test of the ZAP I/O section also indirectly tests memory. This is so because input and output instructions cannot be exercised except by a program stored in memory.

Z80 input and output is handled 8 bits at a time. It does not matter whether the external interface configuration is serial or parallel. Data transfer between the central processor and I/O is 8 bits parallel and basically occurs as follows.

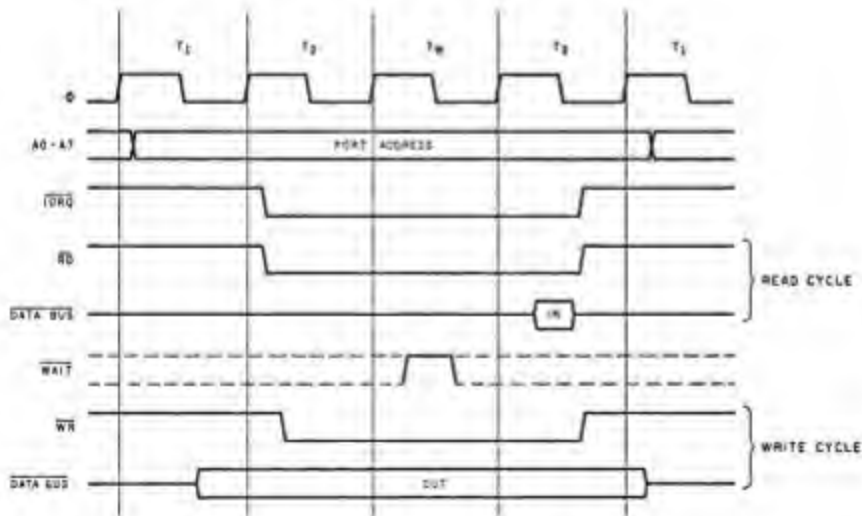


Figure 4.36 A timing diagram of input or output cycles for the Z80.

Output Instruction

OUT(n), A

When this instruction is executed, the contents of the accumulator A are placed on the data bus and written into device n. The address of device n is located on address lines A0 thru A7.

If the accumulator contains 40 hexadecimal when the instruction OUT 23, A is executed, 40 hexadecimal will be written into the peripheral device (also called "port number") decoded as 23 hexadecimal.

While there are other more complicated output instructions available in the Z80 instruction set, they all pass data through the data bus to the external device. Because the data bus is used for transfer of information between the central processor and memory as well as I/O, the computer must be allowed to continue executing its program. Data cannot remain on the data bus waiting for the peripheral (the central processor can be made to do this but such abstract configurations would be confusing at this time). The data is valid for only a few clock cycles and must be stored if needed for a longer period.

Figure 4.37 diagrams a typical 8-bit storage register. It consists of 8 individual storage elements with a common "store enable" input. In its simplest form, the single storage cells can be D-type flip-flops such as shown in figure 4.38. Input data (ie: the data bus) is attached to the D input lines and is only clocked onto the output lines (Q and \bar{Q}) during an I/O write strobe. Using 7474s would require 4 chips for an 8-bit word. A better method is to use the improved circuits of figure 4.39.

Input Instruction

IN A, (n)

When this instruction is executed, the data from the selected port (n) is placed on the data bus and loaded into the accumulator.

If the subject external device reads 10 hexadecimal when the instruction IN A, 20 is executed, the value 10 hexadecimal read from device number 20 hexadecimal would be loaded into the accumulator.

There are other more complicated input instructions but as was the case with output instructions, the route for all data is still the data bus. To keep the data bus from being dominated by a single device attached to it, all input devices (ie: the output from them) must be three-state. This can be accomplished either by using interface logic such as UARTs and peripheral interface adapters that are designed to be three-state, or by adding three-state input buffers such as illustrated in figure 4.40 (the block diagram of the typical 8-bit, parallel-input port).

Whatever is on input lines B₇ thru B₀ during an I/O read instruction will be directed to the central processor. Using these direct read instructions there is no interaction between the central processor and the external hardware attached to the input port. Additional logic is required to coordinate the exact timing between the computer and an external peripheral. The solution is called "handshaking." Such a capability requires either more sophisticated input port hardware, connection to the central processor, interrupt logic, or additional I/O ports to coordinate the timing.

Checking out the basic ZAP hardware is best accomplished by using the least complicated hardware. A simple input port is illustrated in figure 4.41 and consists of 2 quad three-state buffers. Should there be any brave experimenters who wish to have full handshaking on I/O ports or need more than the 8 mA output drive capabilities of a LSTTL device, input and output ports can easily be configured using Intel 8212s. The specifications described in Appendix C5 demonstrate its versatility.

Input/Output Checkout

Ultimately, ZAP could have a keyboard, RS232 serial CRT terminal, audio cassette interface, and analog, as well as digital I/O capabilities. Trying to attach all these pe-

ipherals together and checking everything simultaneously is a monumental undertaking. A more methodical approach is to construct the minimum hardware and software that proves operational and then build upon it. That is the route taken thus far.

With the exception of memory, we have attempted to eliminate any potential problems by static testing where possible. The simple I/O devices of figures 4.39 and 4.41 lend themselves easily to this situation. To test I/O fully requires one input port and one output port. It should be wired as shown in figure 4.42. Only port 0 need be connected at this time. The additional circuitry included in this diagram can be ignored. Only ICs 21 thru 23 are of concern presently. The other devices are enhancements to the basic ZAP and will be discussed later.

Static Test

With power off, remove all ICs previously installed. Insert ICs 20, 21, 22, and 23. Turn on power. Temporarily ground \overline{DSOWR} and \overline{DSORD} . This maneuver, impossible under direct computer control, allows data bus access to both input port 0 and output port 0 at the same time. With the two ports connected in this manner applied input data should be available immediately at the output port. With the input lines of ICs 21 and 22 open and power applied, the outputs of IC 23 should be at a high level. Sequential grounding of input lines B_4 thru B_7 should be reflected on lines B_4 thru B_7 of IC 23. A final test is to disconnect the temporary ground on \overline{DSOWR} while one of the input lines of IC 21 and 22 is grounded. The logic 0 output of IC 23 should remain low even when the input line is no longer grounded. The result is that the data is "latched." It will remain until updated by another write strobe.

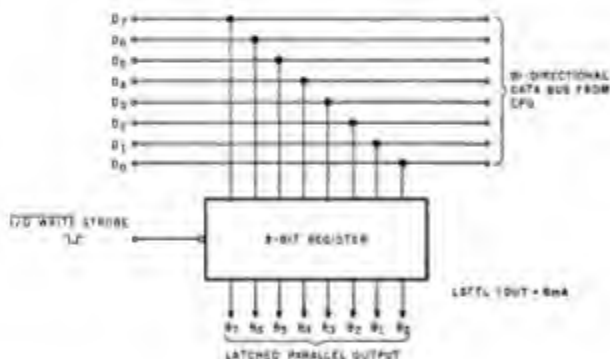


Figure 4.37 A block diagram of a typical latched parallel output port configured with an 8-bit storage register.

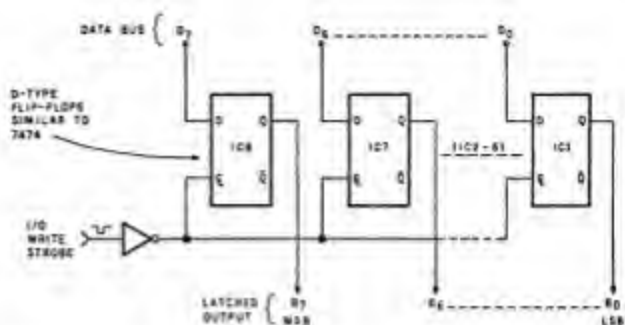


Figure 4.38 A block diagram of a latched parallel output port using D-type flip-flops as a storage register.

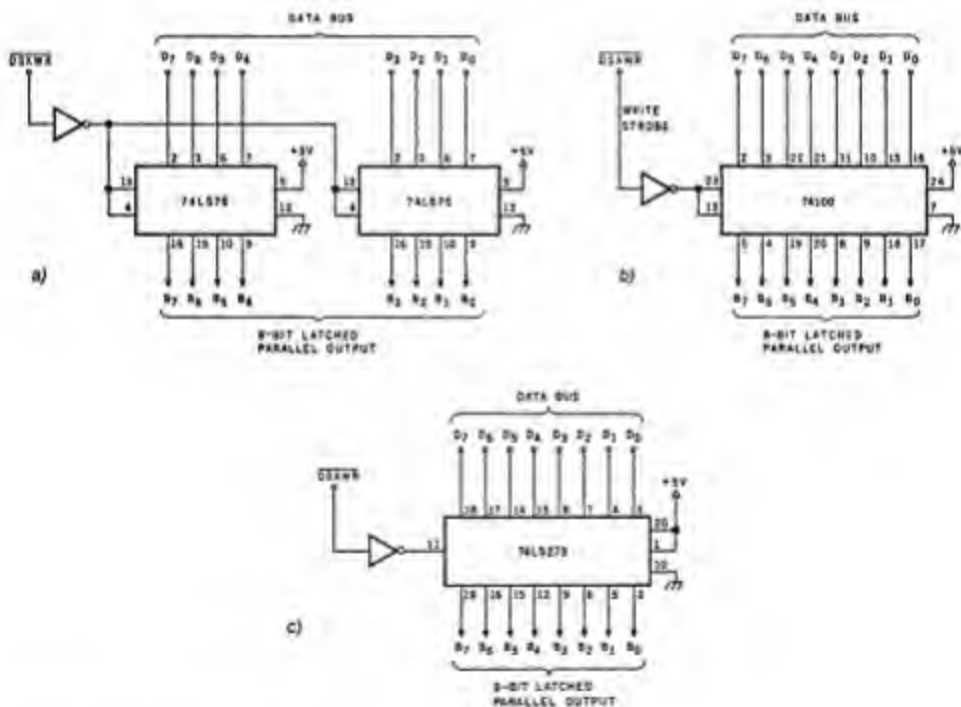


Figure 4.39 Schematic diagrams of 8-bit latched parallel output ports.

- Using two 4-bit LSTTL latches.
- Using a traditional 8-bit TTL latch. Note that non-LSTTL devices can be substituted but care should be taken to observe the total bus loading.
- Using a newer 8-bit LSTTL latch.

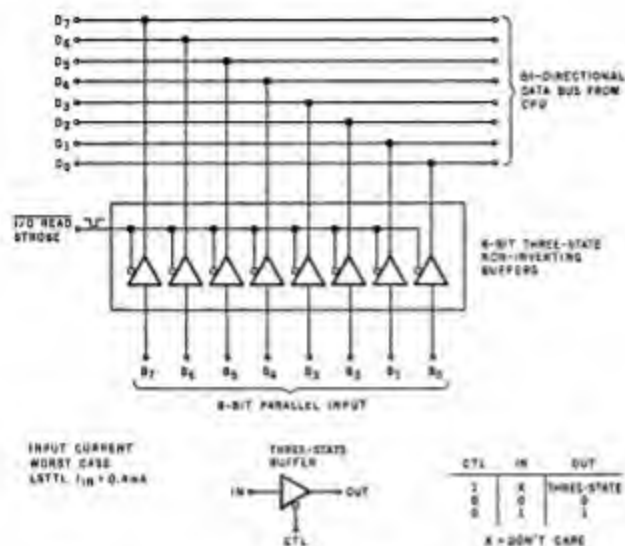


Figure 4.40 A block diagram of a typical 8-bit parallel input port.

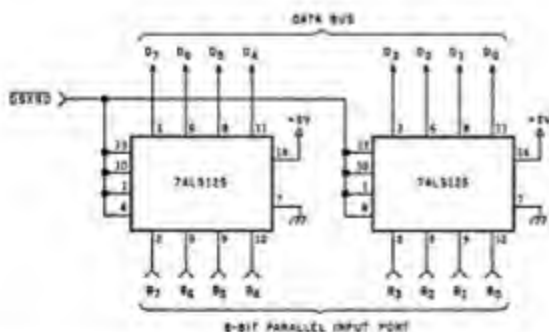


Figure 4.41 A schematic diagram of an 8-bit parallel input port for the ZAP computer.

V. Dynamic Checkout of the Basic Computer

All systems, with the exception of memory, should have successfully passed the static checkout procedures. The memory wiring should be checked for continuity. Because ZAP has no front panel or indicator (unless you wish to add one), the full system can only be tested by executing a program that dynamically exercises all the system hardware. This is easier than it sounds. For the computer to output a number to a specific port address, the central processor must be operational and have reset properly to execute the instruction. The memory read must work or the central processor wouldn't know what to do. The memory and I/O decoding must work for the data stored in memory to arrive at the right output port. And finally, for the data to be read at the port, the output port must function as well. In short, if you can execute a program, the computer works.

We can make the process simpler by using the fewest program steps possible and by initially eliminating the necessity for programmable memory. Remember, ZAP has both EPROM and programmable memory. With no monitor or front panel, programmable memory cannot be loaded directly to run a test program. The test program must be already loaded in ROM (in our case EPROM). By carefully selecting the instructions used in the test program, programmable memory can be left out entirely when we run the first test. Why complicate matters by having more hardware than is necessary?

Few instructions are required to test the operation of the processor, reset, memory and I/O. Usually the central processor either works or it doesn't. Central processor failure is rarely a case of one of the instructions executing improperly. If ZAP can read in data at port 0 and output the same value to output port 0, we can assume it all works. For the data to reach output port 0, it must travel through the central processor (assuming you have removed the temporary grounds on the I/O strobe lines) under program control.

Such a test program is:

	OCTAL	HEXADECIMAL	
IN A, 0	333 000	DB 00	read port 0 in
OUT 0, A	323 000	D3 00	write to port 0 out
JP NN	303 000 000	C3 00 00	jump to beginning

This 7-byte program will read input port 0 data into the accumulator and then write this same data to output port 0. The jump instruction will cause the program to repeat this action continuously. The program requires no programmable memory to store either intermediate data or the stack pointer. Because only the accumulator is affected, the 7-byte program can be completely contained in ROM. In this case, ROM can be either a 2708 EPROM programmed manually as described in Chapter 7 or a simulated ROM as shown in figure 4.30. If you use a simulated ROM, it may be necessary to reduce the 2.5 MHz clock rate to compensate for the capacitance of the external circuitry. Figure 4.30 also includes an output to port 5 that tests a data display to be added later. Rather than rewrite the EPROM or rewire the pseudo-ROM, you may wish to add this instruction now.

The final test of the basic ZAP is to exercise a program that uses both programmable memory and EPROM. Again, the philosophy is that if it can store and retrieve 1 byte from programmable memory, then all 1 K of that bank should work. A slightly longer program is used this time. The following program is stored in EPROM and the programmable memory is used by the central processor to store the stack:

	OCTAL	HEXADECIMAL	
LD SP, nn	061 000 006	31 00 06	set stack pointer to middle of bank 1 programmable memory
IN A, 0	333 000	DB 00	read port 0 input
CALL TEST	315 014 000	CD 0D 00	call program test
OUT 0, A	323 000	D3 00	write data to port 0 out
JP nn	303 000 000	C3 00 00	jump to beginning
TEST RET	311	C9	return to main program

When assembled, the 14-byte program would be loaded as follows (in hexadecimal):

Location	Program
00/00	31 00 06
03	D8 00
05	CD 0D 00
08	D3 00
0A	C3 00 00
0D	C9

The operation of this program is similar to the previous example. A byte is read from input port 0 and then read back out to output port 0. In between these operations there is a call to a subroutine that is just a return instruction. When the call is executed, the location where the program is to resume operation after the call is put on the stack in programmable memory. At the conclusion of the call (the return instruction), the address is popped off the stack and placed in the program counter so that the program can resume where it left off. The only way for the input data from input port 0 to get to output port 0 is for this call to be executed properly. Of course, this requires that programmable memory work properly.

Many other programs that would further enhance the diagnostic checkout procedures can be written. In my experience, however, if it executes these two programs, you can count on everything running.

Once these milestones are reached, the experimenter has a truly operational computer. The next step is to expand this basic unit and make ZAP somewhat more versatile by adding address and data displays, a hexadecimal keyboard, a serial interface, along with an operating system that coordinates the activities of these peripherals. While the present system is a computer, these additions are necessary to move beyond an experimenter's breadboard project.

CHAPTER 5

THE BASIC PERIPHERALS

Once the basic ZAP computer has been constructed and tested, we are ready to add a few necessary peripherals that will greatly increase the system's utility. External peripherals facilitate the input and output capabilities of the computer. They include such items as printers, cathode-ray tubes (CRTs), tape drives, and disks. Peripherals of this magnitude, however, are usually used on larger systems. For our Z80-based ZAP, useful peripherals include a keyboard to ease data and program entry; a visual display to allow the computer to indicate a logical conclusion in readable form; a serial communications interface, which allows ZAP to "talk" to another computer; and an interface to an audio cassette mass storage device. These four ingredients are the difference between an experimental breadboard and a useful personal computer.

The keyboard can be either a small keypad for limited data entry or an alpha-numeric "typewriter"-style ASCII (American Standard Code for Information Interchange) keyboard for text editing and high-level language programming. The visual display could range from a hexadecimal LED readout to a full 24-line by 80-character CRT terminal. The serial port, in conjunction with the audio cassette interface, could be used to cold start the computer and load application programs.

As with the previous circuits in this book, I've tried to provide various alternative designs so that you, the builder, may construct a truly personal system. Each of the four peripheral devices will be explained in detail and numerous design examples will be provided; both limited function hexadecimal input and full ASCII keyboards will be addressed. In the case of the visual display, we will discuss a rudimentary LED octal and a hexadecimal readout for ZAP. For more sophisticated visual interaction, a CRT terminal is required. Because this unit is much more complicated than a keyboard or an LED display, an entire chapter has been dedicated to it. My basic premise is to start with the essentials, provide a thorough understanding of their applications, then move to more complex, more useful add-ons.

The expansion of the basic ZAP into an interactive microcomputer system requires the addition of a software program to synchronize and exercise the new peripherals. This software is called a monitor and is discussed in a later chapter. Peripherals merely provide the means for added data entry and display capability.

1. KEYBOARDS

The only way the Z80 can communicate to an external device is through the input/output bus structure previously described. (While more esoteric methods such as direct memory access exist, they will be ignored for the present.) When the processor wishes to signal the user that an event has occurred, it can do so by changing the output level on one bit of a parallel-output port. For example, the end of program execution can be designated by bit 7 on port 0 going from a logic 0 to a logic 1. Using this concept, 8 separate elements could be individually designated and controlled from the 8 bits of output provided on the single "basic ZAP" port.

Information input is just as simple. The numbers 0 thru 7 could correspond to 8 switches on the 8 input bits of port 0. This is shown graphically in figure 5.1. When

bit-7 switch is pressed, grounding the input, the logic level transition can signify a numeric entry of 7 to the computer; many microprocessor applications require only these few bits of I/O. A traffic light controller, for example, with a single red, yellow, and green light would need only three bits of output.

The program to control the lights would have been written, assembled, and programmed into some type of non-volatile storage. However, ZAP must interact with a human operator in such a way that programs can be developed and tested. The major difference between the traffic light controller and ZAP would be the peripherals and not the microprocessor's capabilities.

In our example, we could put 8 switches on an input port. To enter information, we have only to write a short program that reads the data on port 0 into the accumulator and then stores or acts upon it. The chapter on monitor software will address these manipulations, but one problem must be solved first: synchronizing peripherals to the computer.

How does the computer know when the data on the switches is or is not valid? And, could we make a timer in software or hardware that reads the port every second, on the second? Can you, for example, see yourself trying to flip all the switches in time or to make the computer wait?

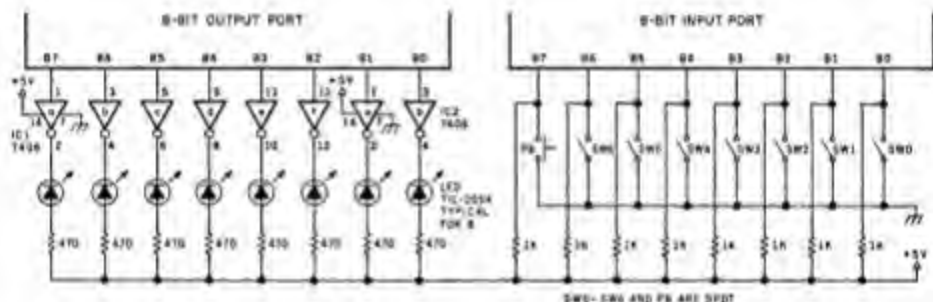


Figure 5.1 A parallel input/output interface with LED readout and switch input.

The most popular method of synchronizing a peripheral that has slow data input to a computer with fast program execution is to use "data ready" strobe pulses. (Interrupts may also be used but they involve complicated programming and will not be considered here.) The program is written to read and check the logic level of one bit only. By substituting a push button for one of the eight switches, say bit 7, we can simulate the strobe. To accomplish this, first set data on the other seven switches; then, with the program sitting in a loop checking bit 7, press the push button to generate a logic transition. The program, sensing that a "data ready" strobe is present, reads in the entire port and uses the other 7 bits of data.

Frequently, it is not practical to limit ourselves to just 7 symbolic interpretations when using 7 bits of input. A more logical approach is to code the input and let the 7 bits represent up to 128 individual symbols. The choice between a coded versus a straight parallel input is governed by the application. If the computer is part of a burglar alarm, with each input bit representing a door or window switch, then it is important to know individual and simultaneous bit transitions. In this application, it is necessary to have parallel signal input. On the other hand, alpha-numeric entry from a typewriter keyboard is by nature serial, one letter at a time. Therefore, nothing is gained by using 128 parallel input bits for a 128-key keyboard. A 7-bit code is more cost-effective.

The most widely used keyboard code is ASCII (American Standard Code for Information Interchange). Appendix B lists the code and the characters it represents. Any homebrew keyboard should reflect this coding to be compatible with commercially available software such as BASIC.

There are a number of methods that can be used to generate suitable key codes. Figures 5.2 and 5.3 reflect hardware and software approaches, respectively. The block diagram outlined in figure 5.2 is a hardware scanning system suitable for a 64-key keyboard. A 6-bit counter progressively enables each column while scanning all rows in each step. Should any key be pressed, a logic 0 will be routed through the 8-input multiplexer to the scan control logic. This signal is used to generate a key-pressed strobe (also called data-ready strobe) to the computer. The row and column address lines from the counter are read and indicate the binary matrix address of the pressed key. Compatibility with the ASCII code is simply a matter of placing the proper key at the correct address within the matrix.

Another suitable encoding method is outlined in figure 5.3. This technique, which uses software logic to scan the matrix, should be used only when computer program execution speed is not critical. While reducing the circuitry to one chip, the trade-off in this approach requires both an input and output port. It functions in the same way as figure 5.2. The computer sets a 4-bit column counter code on the decoder. Then it searches the parallel input port for the row with the logic level 0 signifying a pressed key. While this may seem to be an easy way to decode 128 keys, there are certain software considerations.

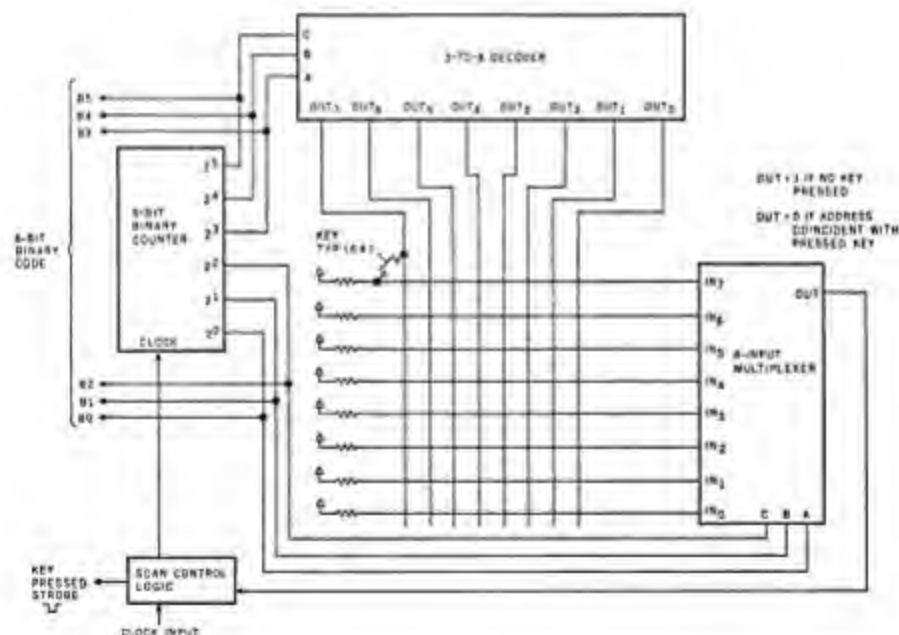


Figure 5.2 A matrix keyboard scanner for a 64-key keyboard.

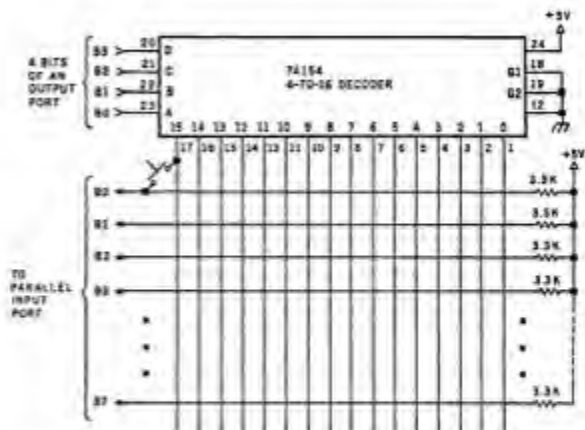


Figure 5.3 A software-driven 128-key encoder circuit.

The key-pressed or data-ready strobe in any keyboard serves two purposes: it signifies that data is present and ready, and it is timed so the strobe is not generated until after a mechanical debounce time period has elapsed. The reason for the delay is obvious. Remember, these microprocessors can execute 200,000 instructions a second. A program written to look for a strobe and read the data would run a hundred times on a single keypress because of contact bounce. The mechanical making and breaking of the contact could appear like 100 data-ready strobes if we aren't careful. A true data-ready strobe is not generated until after a debounce time-out and then it should be fast-rise-time (<200 ns) pulse with a rate exceeding the cycle time of the computer. The duration of the pulse should be long enough to allow the scanning program to catch it even if it is off doing some other task, and short enough so that the central processor doesn't see the same strobe twice.

There are two techniques to combat the problem of strobe duration. One is to set a flip-flop with the rising edge of the strobe and tie the clear line of the flip-flop to an output bit. After reading in the data, the program can clear the "data-ready" condition by resetting the flip-flop. This is usually employed in cases where the response time to a keyboard or other device is variable. This method also guarantees that an event will be registered and not missed due to time delays. Of course, most keyboard encoders do not latch their output data. If a key is released, even if the strobe has been set in a flip-flop, no data will be present when the computer reads the keyboard. There are ways to get around this but they all involve additional hardware.

Usually the experimenter's problem is reading a strobe twice rather than not waiting long enough to acknowledge it. Instead of using a hardware flip-flop, most programmers employ a software flag, the second technique in dealing with strobe duration. When a key-pressed strobe is sensed, the program sets a flag in a memory location, reads the data, then checks the strobe again. If the strobe is high, the flag is checked and the data is not read. Only when the strobe returns to a logic zero is the flag reset, enabling data input the next time.

It's not easy to construct keyboard encoders for 64- or 128-key ASCII keyboards. It's simpler to use a commercially available, scanning, read-only memory encoder such as the one documented in Appendix C6.

As far as ZAP is concerned, it is important to learn to walk before we run. Most people would consider ZAP to be a learning tool that could be eventually expanded into a full-blown microcomputer system. A full 128-key ASCII keyboard could prove to be as expensive as the entire ZAP computer. To minimize expense and retain the experimen-

The keyboard required to support the ZAP software monitor has 19 keys. The encoder in figure 5.4 is a combination scanner and hard-wired parallel output. Encoding depends upon the particular key pressed. The hexadecimal keys 0 thru F are sensed through a multiplexed scanner, IC 2 and IC 3. As IC 2 counts, it sequentially places a logic 0 on each of the 16 output lines of IC 3. If any key is pressed, that low level is routed back to IC 4 and stops the clock. The counter is then locked on the address of the particular key being pressed. The same action that stops the clock also triggers a one-shot IC 5 which generates a key-pressed strobe. The output lines B0 thru B3 will contain the binary value of the pressed key while bit 7 is reserved for the strobe. The three function keys are directly tied to input bits 4, 5, and 6. Three sections of IC 1 serve to dampen contact bounce. The EXEC and NEXT are tied in so they will generate a key-pressed strobe when activated. Because the shift key is always used in conjunction with another key, it is not connected to the strobe circuit.

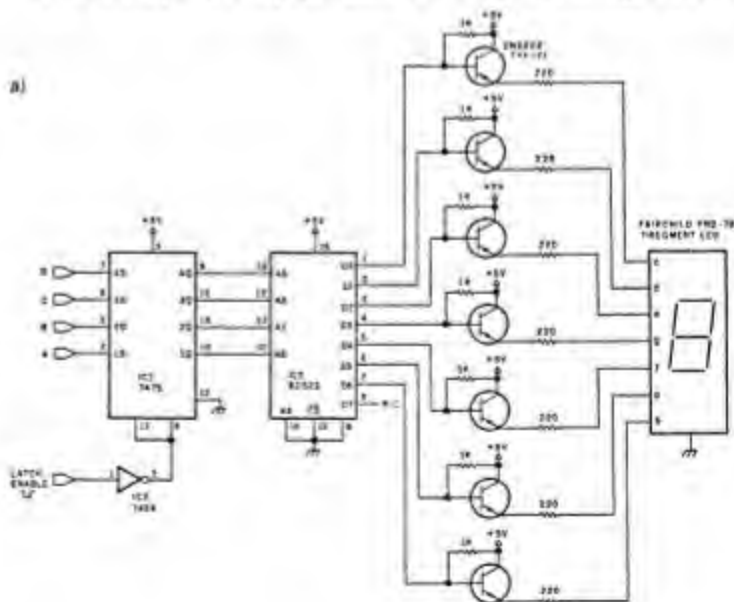
It is important to recognize that the coding of this 19-key circuit is not ASCII. An ASCII keyboard cannot be used directly with the software monitor outlined in this book, unless you use only those ASCII keys that correspond to the coding of figure 5.4, or rewrite the software monitor to accept ASCII rather than binary codes for each key.

II. ADDING A VISUAL DISPLAY

Once a keyboard has been added to ZAP, we are ready for program development. The other key ingredient is a visual display that allows the programmer to examine instruction statements and data. The least costly configuration is an LED display, preferably hexadecimal because the software monitor is written that way. For the octal di-hards, I've also included an octal display.

Hexadecimal displays may seem a trivial addition to an expensive computer system, but it is sometimes these little helpful add-ons that make program debugging easier. I don't intend that it should replace a CRT, but it's a necessary tool when debugging a program and a necessity for using the ZAP monitor. It will never replace a stepper or a break-point-monitor program, but it's great to display keyboard or I/O data quickly with a single output instruction.

There are many ways to display hexadecimal on a 7-segment LED. Figure 5.5 is an



b)	INPUT CODE	82S23 PROGRAM	7-SEGMENT DISPLAY
	D C B A	D7D6D5D4D3D2D1D0	
	0 0 0 0	0 1 1 1 0 1 1 1	0
	0 0 0 1	0 1 0 0 0 0 0 1	1
	0 0 1 0	0 1 1 0 1 1 1 0	2
	0 0 1 1	0 1 1 0 1 0 1 1	3
	0 1 0 0	0 1 0 1 1 0 0 1	4
	0 1 0 1	0 0 1 1 1 0 1 1	5
	0 1 1 0	0 0 1 1 1 1 1 1	6
	0 1 1 1	0 1 1 0 0 0 0 1	7
	1 0 0 0	0 1 1 1 1 1 1 1	8
	1 0 0 1	0 1 1 1 1 0 0 1	9
	1 0 1 0	0 1 1 1 1 1 0 1	A
	1 0 1 1	0 0 0 1 1 1 1 1	b
	1 1 0 0	0 0 1 1 0 1 1 0	C
	1 1 0 1	0 1 0 0 1 1 1 1	d
	1 1 1 0	0 0 1 1 1 1 1 0	E
	1 1 1 1	0 0 1 1 1 1 0 0	F

Figure 5.5 A possible method for a hexadecimal latch/decoder/driver using a standard 7-segment LED.

- a) This entire circuit would be needed to replace one HP7340. \overline{CS} on the 82S23 can perform the blanking function.
 b) The program for the 82S23 (IC 2).

example of the usual brute force method using a PROM as a hexadecimal decoder. (A method of programming the 82S23 was described in the article in the November 1975 issue of *BYTE* magazine entitled "A Versatile Read-Only Memory Programmer," if you choose to use this circuit.)

However, this approach uses an excessive number of components and most people would not want to program a PROM. One alternative is to allow the computer to perform the decoding and drive the 7-segment display through the transistors directly from a latched 8-bit output port. Another way puts additional logic around a standard 7-segment decoder driver for the extra requirements. The former case necessitates a computer program while the latter can involve as many components as figure 5.5.

Fortunately, there is a product on the market that can solve the problem. It is the HP7340 hexadecimal LED display (from Hewlett Packard; equivalent displays are available from other manufacturers). These hexadecimal digits depart from the standard 7-segment format by using dots instead of bars and being capable of displaying a capital "B" and "D" in hexadecimal. This is accomplished by controlling the corner dots, which gives the appearance of "rounding." This ability discriminates a "B" from an "8" or a "D" from a "0." There are 16 distinctly different characters.

An additional feature of the HP7340 is that each display circuit contains a 4-bit latch and decoder/driver. This allows the display to be attached directly to the data bus. The result is a single 8-pin hexadecimal display that successfully accomplishes the function of all the circuitry of figure 5.5. The specifications of the individual pins are given in figure 5.6.

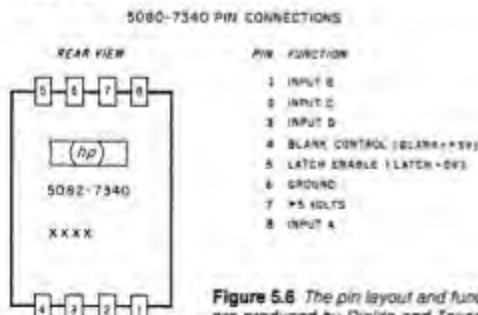


Figure 5.6 The pin layout and functions for the HP7340 BCD to hexadecimal display. Similar displays are produced by Discrete and Texas Instruments.

Figures 5.7 and 5.8 demonstrate how the HP7340 can be configured to function as a 2-digit hexadecimal output port or a 3-digit octal port. An 8-bit latch is not required because it already contains one. The HP7340s can be attached to the data bus as simply as any other parallel output port and are strobed from the chip-select decoder outlined earlier in the section on I/O decoding.

To utilize the software monitor properly, 6 hexadecimal displays (separated into 3 single byte displays) are necessary. Three bytes are required to display a particular H and L address and the data contents of that location. The 6 hexadecimal displays should have the following decoded strobes:

Output Port #	Logic Line	Display Parameter	IC#
5	$\overline{DS5WR}$	MSD address field	30, 31
6	$\overline{DS6WR}$	LSD address field	28, 29
7	$\overline{DS7WR}$	data field	26, 27

MSD — Most Significant Digit

LSD — Least Significant Digit

A more complete description of each display function is described within the monitor section, and a completed schematic showing how the 6 displays are attached to the data bus is illustrated in figure 5.9.

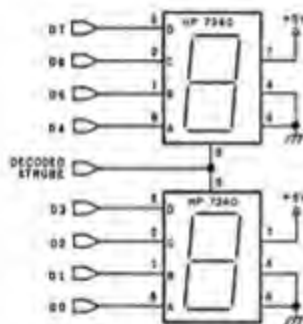


Figure 5.7 An HP7340 hexadecimal latch/decoder/driver display.

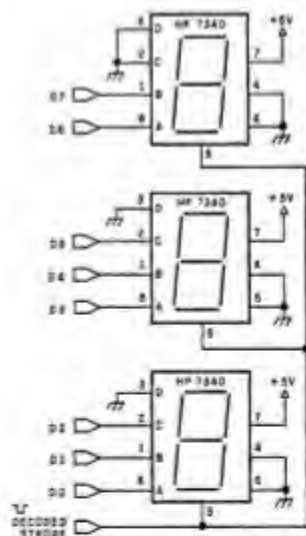


Figure 5.8 An HP7340 octal latch/decoder/driver display. The HP5082-7300 can be substituted for the HP5082-7340 in octal display applications. The HP7300 displays numerics only.

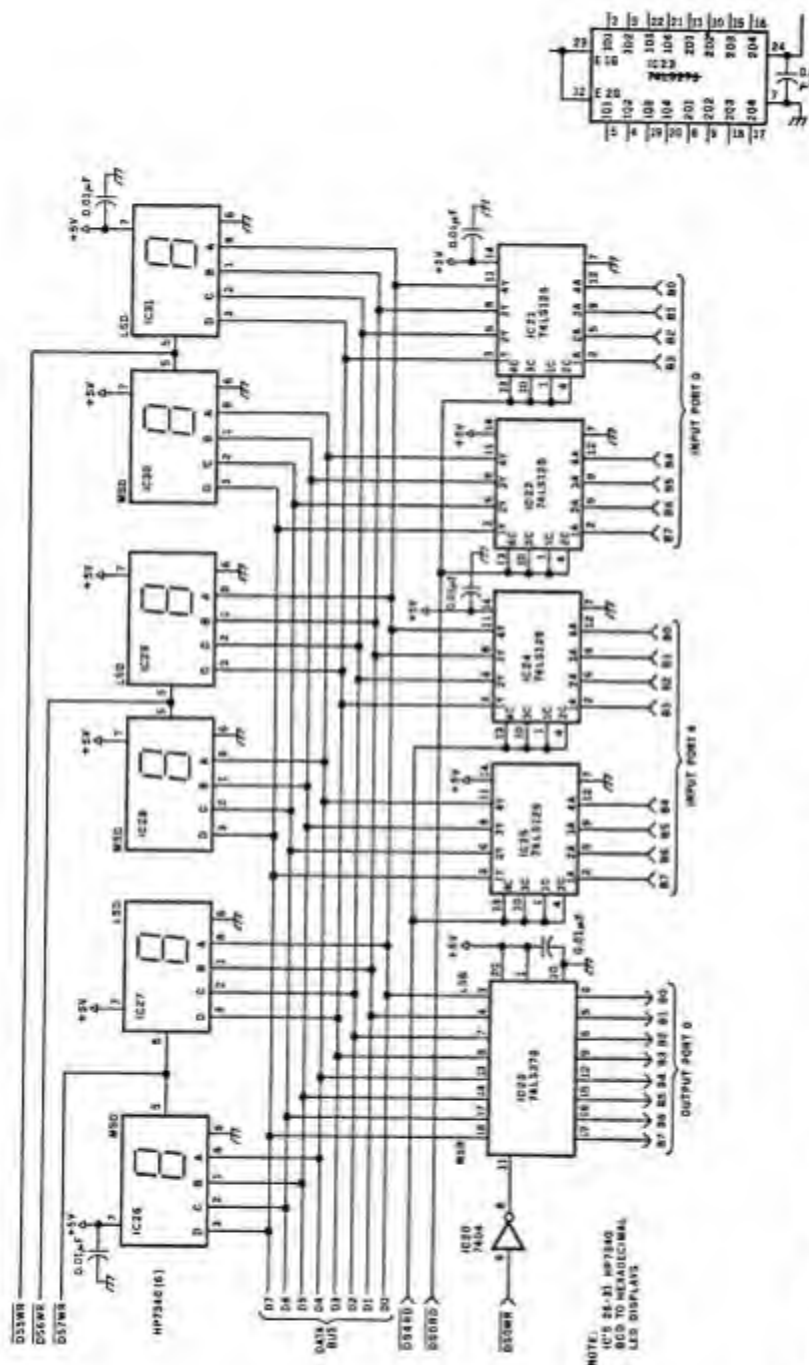


Figure 5.9 A schematic diagram of the completed hexadecimal LED display.

III. SERIAL INTERFACE

A serial communication capability is not absolutely necessary to make ZAP work, although the software monitor supplied in this book supports a serial interface.

First a word about concept before we pursue the design details. Why would ZAP need to communicate? When we discuss the serial cassette interface, you will understand that there are more advantages to it than appear presently. If future expansion is in mind or commercially made peripherals such as a CRT or printer are ever added, their interface will most likely be serial.

This last sentence is significant. Realize that I said nothing about communicating with another computer. While talking to another computer over telephone lines requires a serial link, in general, standard peripherals such as CRTs and printers also "talk" serially. Therefore, by designing a serial port to accommodate a printer, we also gain the ability to talk with another computer.

Communication is simply the transfer of information from one device to another. In the case of a CRT display unit, the computer sends character information for screen display while the keyboard relays the user's input to the computer. Each end of the full-duplex communication line must have a transmitter and a receiver. In both cases, the information being transferred is ASCII data probably consisting of a 7-bit code and, in some cases, an additional parity bit for error checking. This 7-bit data (ignoring the parity bit) will appear on the lines of a parallel port. These 7 lines plus a ground reference and a strobe (remember we have to tell the receiver when the data is valid) can be brought out to the CRT input. Keeping that as a dedicated line from the computer to the CRT, we now want a similar line between the keyboard output and an 8-bit parallel port on the computer. This requires an additional 9 lines. To further complicate matters, let's separate the terminal and the computer by 300 to 400 feet, as might happen in some commercial computer systems. The result is that 400 feet of 18 lead (17 if you combine ground references) cable will cost more than the terminal. Also realize that the TTL parallel output should not be used to drive lines longer than 20 feet without special buffers/drivers; otherwise data errors could occur.

The solution to this costly wiring problem is to use serial rather than parallel communication. The parallel data is converted to serial and sent one bit at a time down a single twisted pair wire. If buffers/drivers are needed for long distances, less are required with the serial approach. Specially encoded "start" and "stop" bits included in the serial transmission notify the receiver that valid data is being sent. For the above example, only two pairs of wire are needed to perform "full-duplex" interaction (see figure 5.10). In "half-duplex" mode this can be reduced to a single twisted pair, but synchronization of the shared communication line is more complicated. All serial transmission references I shall make will be limited to full-duplex operation.

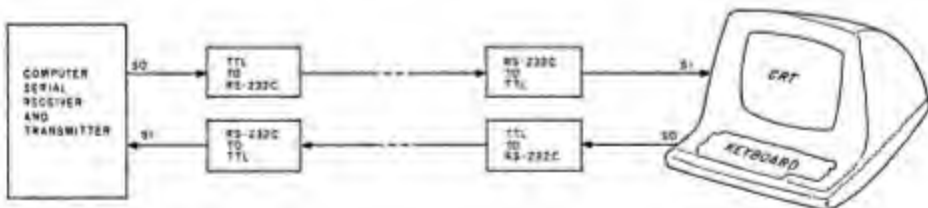


Figure 5.10 A block diagram of a full-duplex RS-232C communication link.

Now that we agree that the communication should be serial, how do we accomplish the parallel to serial conversion? The answer is a device called a UART (Universal Asynchronous Receiver/Transmitter). Appendix C7 gives the specification information for the SMC COM2017 UART which is equivalent in function to the AY-5-1013A (General Instruments). To minimize power supply requirements, a single +5 V AY-3-1015 or TR1602 (Western Digital) can be substituted as I have done. The only change from the specification sheet is that pin #2 is no longer tied to -12 V.

A UART's internal structure consists of a separate parallel-to-serial transmitter and serial-to-parallel receiver joined by common programming pins. This means that the two sections of the UART can be used independently, provided they adhere to the same bit format that is hard-wire or software selectable on the chip.

The transmission from the computer to the CRT is done asynchronously and in one direction only. The computer likewise receives data directly from the keyboard through a dedicated line. As far as the computer is concerned, after reversion to parallel in the UART, this input device is communicating parallel data.

Actual data transmission follows the asynchronous serial format (illustrated in figure 5.11). Using the keyboard as an example, when no data is being transmitted, the data line is sitting at a mark (or "1" level) waiting for a key-pressed strobe. A key-pressed strobe is a 1 to 5 ms positive pulse (it can be as short as 200 ns) indicating that a keyboard key has been pressed, and that an ASCII code of that key is available for transmission. This key-pressed strobe, which is attached to the data strobe of the UART, causes the ASCII data to be loaded into a parallel storage buffer and starts the UART transmission cycle. The serial output will then make a transition from a 1 to a 0. This mark-to-0 start bit is 1 clock period long and indicates the beginning of a serially transmitted word. Following the start bit, up to 8 bits of data follow, each data bit taking 1 clock period. At the conclusion of the data bits, parity and stop bits are output by the UART to signify the end of transmission. If another key is pressed, the process repeats itself.

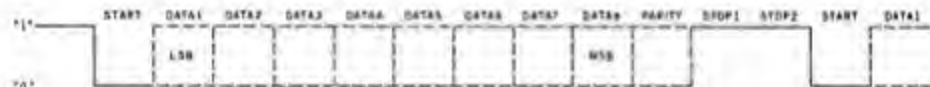


Figure 5.11 A single data byte as it is transmitted in asynchronous serial format.

On the receiving end, the UART is continuously monitoring the serial input line for the start bit. Upon its occurrence, the 8 bits of data are slipped into a register and the parity checked. At the completion of the serial entry, an output signifying data available is set by the UART and can be used as an input strobe to the computer. The UART will not process additional serial inputs unless the data available flag is acknowledged, and the data available reset line is strobed. Actual transmission can include or exclude parity, have 1 or 2 stop-bits, and data can be in 5- to 8-bit words. These options are pin selectable.

The following is a pin function description for the AY-5-1013, COM2017, or AY-3-1015.

Pin #	NAME	SYMBOL	FUNCTION
1	V _{cc} Power Supply	V _{cc}	+5 V Supply
2	V _{ee} Power Supply	V _{ee}	-12 V Supply (not connected on AY-3-1015)
3	Ground	GND	Ground
4	Received Data Enable	RDE	A logic "0" on the received enable line places the received data onto the output

5 thru 12	Received Data Bits	RD8 thru RD1	lines. These are the eight data output lines. Received characters are right justified; the LSB always appears on RD1. These lines have three-state outputs.
13	Parity Error	PE	This line goes to a logic "1" if the received character parity does not agree with the selected parity. Three-state.
14	Framing Error	FE	This line goes to a logic "1" if the received character has no valid stop bit. Three-state.
15	Over-Run	OR	This line goes to a logic "1" if the previously received character is not read (DAV line not reset) before the present character is transferred to the receiver holding register. Three-state.
16	Status Word Enable	$\overline{\text{SWE}}$	A logic "0" on this line places the status word bits (PE, FE, OR, DAV, TBMT) onto the output lines. Three-state.
17	Receiver Clock	RCP	This line should have as an input a clock whose frequency is 16 times (16X) the desired receiver data rate.
18	$\overline{\text{Reset Data Available}}$	$\overline{\text{RDAV}}$	A logic "0" will reset the DAV line.
19	Data Available	DAV	This line goes to a logic "1" when an entire character has been received and transferred to the receiver holding register. Three-state.
20	Serial Input	SI	This line accepts the serial bit input stream. A marking (logic "1") to spacing (logic "0") transition is required for initiation of data reception.
21	External Reset	XR	Resets shift registers. Sets SO, EOC, and TBMT to a logic "1." Resets DAV and error flags to "0." Clears input data buffer. Must be tied to logic "0" when not in use.
22	Transmitter Buffer Empty	TBMT	The transmitter buffer empty flag goes to logic "1" when the data bits holding

23	Data Strobe	\overline{DS}	<p>register may be loaded with another character. Three-state.</p> <p>A strobe on this line will enter the data bits into the data bits holding register. Initial data transmission is initiated by the rising edge of \overline{DS}. Data must be stable during entire strobe.</p>
24	End of Character	EOC	<p>This line goes to a logic "1" each time a full character is transmitted. It remains at this level until the start of transmission of the next character.</p>
25	Serial Output	SO	<p>This line will serially, bit by bit, provide the entire transmitted character. It will remain at logic "1" when no data is being transmitted.</p>
26 thru 33	Data Bit Inputs	BD1 thru BD8	<p>There are up to eight data bit input lines available.</p>
34	Control Strobe	CS	<p>A logic "1" on this lead will enter the control bits (EPS, NB1, NB2, TSB, NP) into the control bits holding register. This line can be strobed or hard-wired to a logic "1" level.</p>
35	No Parity	NP	<p>A logic "1" on this lead will eliminate the parity bit from the transmitted and received character (no PE indication). The stop bit(s) will immediately follow the last data bit. If not used, this lead must be tied to a logic "0."</p>
36	Number of Stop Bits	TSB	<p>This lead will select the number of stop bits, one or two, to be appended immediately after the parity bit. A logic "0" will insert 2 stop bits. A logic "1" inserts 1 stop bit.</p>
37 38	Number of Bits/ Characters	NB2, NB1	<p>These two leads will be internally decoded to select either 5, 6, 7 or 8 data bits/character.</p>

NB2	NB1	Bits/Character
0	0	5
0	1	6
1	0	7
1	1	8

39	Odd/Even Parity Select	EPS	The logic level on this pin selects the type of parity that will be appended immediately after the data bits. It also determines the parity that will be checked by the receiver. A logic "0" will insert odd parity, and a logic "1" will insert even parity.
40	Transmitter Clock	TCP	This line should have as an input a clock whose frequency is 16 times (16x) the desired transmitter data rate.

The final serial interface configuration is shown in figure 5.12. Because a UART is a three-state device, it can be attached directly to the data bus. Data is written into or read from it 8 bits parallel as any other I/O port manipulation. To the computer, the UART appears as one output and two input registers: status, transmitted data, and received data. As with all data bus manipulations, data transfers are synchronized through decoded strobes. The ZAP software monitor uses three port addresses to coordinate the hardware and software. To be compatible, they should be wired as follows:

Port #	Logic Line	Signal
02 INPUT	DS2RD	READ DATA
03 INPUT	DS3RD	READ STATUS
02 OUTPUT	DS2WK	WRITE DATA

The primary focus of this chapter is the hardware section of the serial interface. When connected directly to the data bus in this manner, there is no way to operate the UART except under program control. Explanation of the protocol and the significance of each UART register can be found in the section on the ZAP monitor.

There are two remaining hardware considerations: data rate and transmission signal level. Data rate can be loosely termed as bits per second and refers to the transmission speed along the twisted pair. Keep in mind that at lower data rates, only 8 of 11 bits of each transmitted word are data; 1 start bit and 2 stop bits are used. While any transmission frequency can be set on a UART, by adjusting the clock rate there are eight frequently used standard asynchronous transmission rates:

110 bps
150 bps
300 bps
600 bps
1200 bps
2400 bps
4800 bps
9600 bps

Using a special data rate generator chip and switch selector network shown in figure 5.12, ZAP can accommodate any of these specific frequencies. In normal operation, most teletypes run at 110 bps, printers such as the DECwriter II at 300 bps, acoustic telephone modems at 300 bps, and video terminals from 1200 to 19,200 bps. As you can see, in theory, we can communicate with them.

Transmission rate is only part of inter-communication prerequisites. A computer could be all TTL level logic while a peripheral used 15 V CMOS. They would be completely incompatible. Therefore, it is necessary to have one additional standard that governs the signal level of the transmissions. The most widely accepted and generally

used standard is EIA RS-232C.

Although TTL levels could be used for communication, they are not suitable for carrying signals more than 10 or 20 feet. The problem stems from the fact that only 2 V separates a logic 1 or 0 rather than speed or drive capabilities. With only 2 V immunity to noise, communication would be susceptible to interference from motors and switches.

An industrial committee agreed to a standard interface to solve this problem as well as to suggest standards for the industry. Modem equipment uses EIA RS-232C. This specification applies not only to the specific voltages assigned to logic 0 and 1, but also to the type of plug, pin assignments, source and load impedances, as well as to a variety of other related functions.

The signal levels of RS-232C are bipolar and use a negative voltage between -3 and -15 V to represent a logic 1 and a positive 3 to 15 V to represent a logic 0. The region between -3 V and $+3$ V helps our noise immunity and is a dead region. Even though $+15$ V and -15 V would provide optimum transmission, $+3$ V and -7 V are also acceptable. However, try to maintain equal bipolar levels over long distances.

The basic ZAP computer requires $+12$, $+5$, and -12 V (-5 V is necessary for the EPROM memory and is derived from the -12 V supply) supplies for operation. We can use the positive and negative supplies to generate RS-232C voltage levels in a number of ways. Figure 5.13 illustrates some RS-232C drivers, and figure 5.14 shows a couple of receiver circuits. One from each selection would have to be attached to the serial I/O pins of the UART for it to have complete RS-232C compatibility.

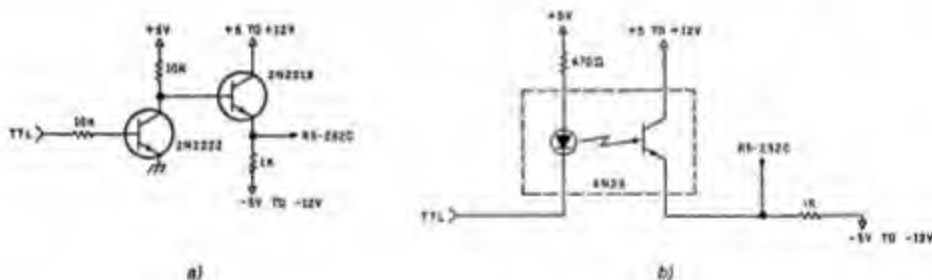
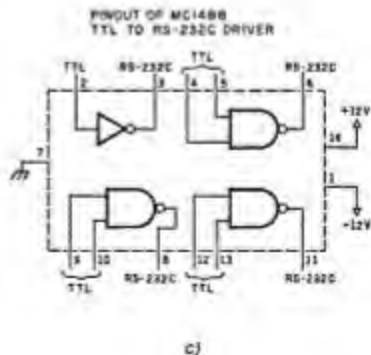


Figure 5.13 TTL to RS-232C drivers.

- Using two transistors as a level shifter.
- Using an opto-isolator as a level shifter.
- Using a standard RS-232C line driver.



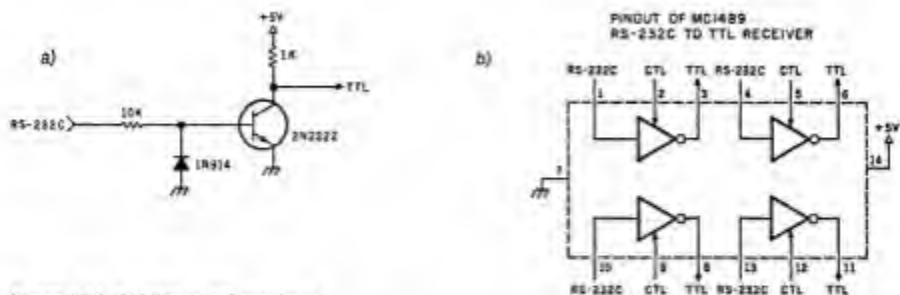


Figure 5.14 RS-232C to TTL receivers.

a) Using a transistor.

b) Using a standard RS-232C line receiver.

IV. CASSETTE STORAGE INTERFACE

The last but by no means least of the enhancements we should add to ZAP[®] is a cassette interface. With the keyboard and display, an operator will be able to write some elaborate programs but, unless they are transferred into read-only memory storage, they will be lost when power is turned off. Of course, the computer's power can be left on constantly. But what if you want to develop a second program that must occupy the same memory address space? The preferable solution is to have some medium that temporarily stores large memory blocks.

In large computer systems, this capability is achieved through hard-disk and 9-track magnetic tape systems. These high-speed, high-volume media are beyond the personal computing budget, but their value in large systems is obvious. A low price, lower performance alternative is an audio cassette storage system.

In general, a cassette storage interface consists of three major subsystems: a serial transmitter/receiver; a hardware assembly that converts serial TTL data so it's audio cassette compatible; and an application program that keeps track of what's going out to tape and can load it back into the correct place. The basic configuration is illustrated in block diagram form in figure 5.15.

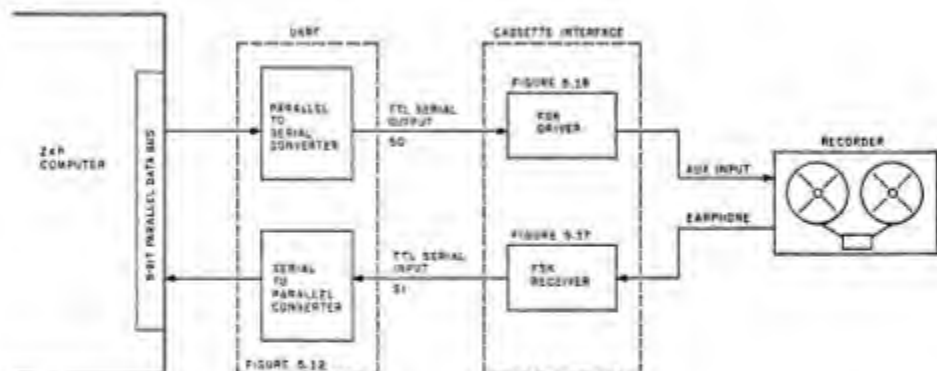


Figure 5.15 A block diagram of an audio cassette storage system.

The serial transmitter/receiver section is nothing more than the UART serial interface which we have already added. With MC1488 and 89 converters on its serial lines, it communicates via a RS-232C. However, if you attach a cassette interface to these lines, it can double as a storage device. An additional benefit is that serial data generated by the UART will offer some compatibility between personal computing systems; standard data rates and standard serial communication protocol will promote this.

The output of the UART is TTL. Even with the RS-232C drivers, the logic output is still a DC level. Because audio recorders cannot record DC, the UART output must be converted in some way. The solution is FSK (frequency shift keying). The TTL output from the UART is converted into audio tones. One frequency represents a logic 0, and a second represents a logic 1.

Figure 5.16 shows a circuit that will produce frequency shift keyed tones. A 4800 Hz reference frequency is derived from the MC14411 data rate generator previously installed. IC 2A and 2B function as a programmable divider chain. With a TTL logic 1 on the input IC 2 divides the 4800 Hz by 2, giving a 2400 Hz output. When the input level is changed to logic 0, it divides by 4, producing a 1200 Hz output. The FSK frequencies are generated at a serial output rate of 300 bps and connected directly to the recorder through the microphone or auxiliary input. (These frequencies and data rate are often referred to as the Kansas City Standard.)

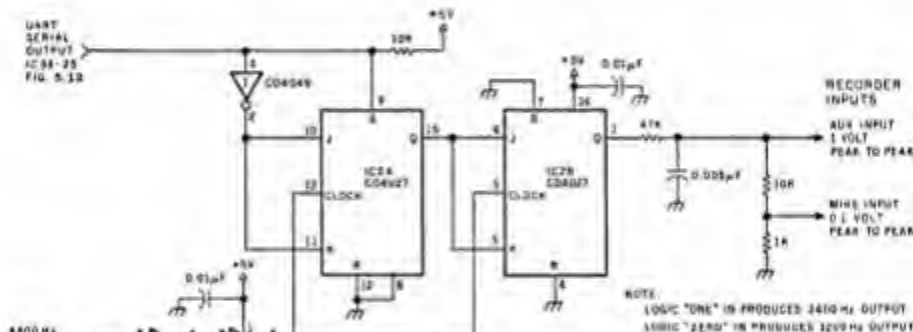


Figure 5.16 A 300 bps serial output driver to an audio recorder.

Getting the recorded tones off the audio tape requires the circuit shown in figure 5.17. In general, it consists of a pair of band-pass filters and a voltage comparator. The recorder is set to an output level of approximately 1 V peak to peak. This level is not critical because it is amplified and limited as it passes through IC 1, IC 2 and IC 3 are band-pass filters with center frequencies of 2400 Hz and 1200 Hz, respectively. The output of IC 1 is fed into both of them, but should be passed by only one. IC 4 compares the outputs of the two filters and generates a TTL logic 1 when a 2400 Hz tone is received and a logic 0 with a 1200 Hz tone. Tuning the interface will be explained later.

The choice of the FSK frequencies and data rate are not left to chance. They are a function of receiver response speed and recorder bandwidth. Most cassette recorders have a frequency response of around 8 kHz. Less expensive units can be as low as 5 or 6 kHz. It is unwise to try to record tones at this upper limit. The center of the frequency range offers more reliability, so the logic "1" FSK tone should be set less than 3 kHz (2400 Hz in our case). In addition, it takes time for the receiver to recognize a particular frequency. The circuit of figure 5.17 takes 2 or 3 cycles to respond. This means that at the low frequency of 1200 Hz, each logic 0 bit will need 3 cycles at 1200 Hz to be recognized.

If we consider a worst case condition of sending all zeros, the transmission rate would have to be slower than 400 bps to be accurately received. The closest standard data rate to this value is 300 bps. Raising the 1200 Hz tone to increase the transmission speed only complicates the filter design the closer it is to 2400 Hz. This interface has been tested at 600 bps but it requires precise alignment to achieve faster speeds. The low frequencies and moderate data rate are chosen specifically to increase the probability of successful construction rather than to compete with high speed data storage systems.

The final point to consider is the software that runs the hardware. The ZAP monitor, as it now stands, does not directly support a cassette interface even though it does handle all the serial housekeeping. Until you write the cassette driver into an EPROM, you will have to type in a short "bootstrap" program. To read the cassette, the logic of the program would follow the flow diagram in figure 5.18.

First, a pointer is set in the H and L registers to designate where the cassette data will be stored in programmable memory and an address where it will end. Next, taking advantage of the serial communication routine in the ZAP monitor, we simply call "SERIAL IN" which returns with a byte of data from the UART. This byte is stored in memory, and the HL register pair is decremented and compared to a predetermined stop address. If not equal, it repeats the process of getting another byte of data.

Storing memory is equally straightforward and is diagrammed in figure 5.19. Again, a pointer is set to the beginning and the memory area to be written to tape. Next, the "SERIAL OUT" routine is called from the ZAP monitor, which sends the byte of data to the cassette. Finally, the pointer is decremented and compared to the end address to see if more data is to be written.

These are relatively easy routines to write and short enough that they may be squeezed into the few empty bytes within the ZAP monitor EPROM. Whatever the case, you will soon realize the versatility and capability that such a simple interface adds to a computer system. The 2 K of programmable memory on the basic ZAP will become resident program space while the cassette will be a potential megabyte file storage system for it.

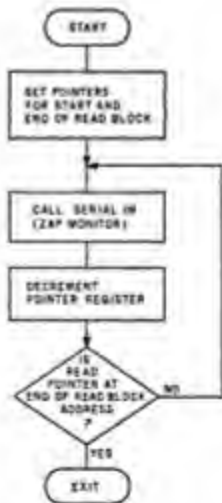


Figure 5.18 A flowchart of software to read a cassette.

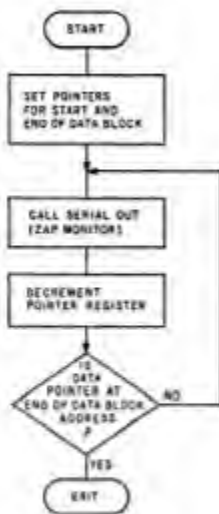


Figure 5.19 A flowchart of software to write a cassette.

TUNING THE CASSETTE INTERFACE

To test the cassette interface, it is necessary first to construct the circuit from figure 5.16. Use a frequency counter to determine that the input to IC 1, pin 5 is 4800 Hz. With no UART installed, the frequency at pin 1 of IC 2b should be 2400 Hz. Grounding IC 2b, pin 1 should change this output to 1200 Hz. In both cases, voltages of 1 and 0.1 V should be present on the cassette auxiliary and microphone inputs respectively.

The receiver uses the frequencies generated by the output section previously described to set the calibration. With the output section set to 2400 Hz, attach a jumper from the output interface to the input of the receiver circuit (figure 5.17). Using an oscilloscope, check that the waveform at IC 1, pin 6 is a square wave of 2400 Hz. Next, with the scope attached to IC 2, pin 6, adjust R1 until the voltage at that point is maximum. Moving the scope probe to IC 3, pin 6, and changing the input frequency to 1200 Hz, repeat the procedure by adjusting R2 until the voltage peaks.

R3 sets the point at which the comparator switches between logic levels when the input frequencies change. The proper way to set this is to use a function generator on the input and set R3 to switch at exactly 1800 Hz. The result should be clean logic level switching at IC 4, pin 6, as the frequency is cycled between 1200 Hz and 2400 Hz. Generally speaking, the comparator setting is not especially critical.

CHAPTER 6

THE ZAP MONITOR SOFTWARE

The function of an operating system is to provide the programmer with a set of tools to help him in developing, debugging and executing a program. In general, the operating system assists the programmer by managing the resources of the computer, and by eliminating his involvement with repetitive machine-code manipulations. Operating systems span a broad spectrum of complexity. Small systems, for example, provide only a rudimentary means for a programmer to enter and read 8-bit data from memory; large systems, on the other hand, can dynamically manage the allocation of all memory and peripherals.

Large systems allocate computer resources to more than one user in a multiprogramming, multitasking, or a time sharing environment. A system of this magnitude far exceeds the capabilities of the computer described in this book. This being the case, what would be a suitable operating system for the ZAP computer? As previously stated, the objective of an operating system is to manage the resources of the computer. The ZAP computer described in the previous chapters, and enhanced with the minimum peripherals, contains the following resources:

- Z80 microprocessor
- 1024 bytes of EPROM memory
- 1024 bytes of programmable memory (2048 optional)
- Nineteen-key keyboard
- Two-character data display
- Four-character address display
- UART for serial I/O

The operating system must provide access to these resources and give the user a way to manage them during execution of programs. The operating system designed for ZAP will include the following facilities and functions:

1. Cold start
2. Warm start
3. Memory display and replace
4. Register display and replace
5. Execute (begin program execution at a designated point)
6. Serial input and output

Each will be explained in detail concerning its functions and program implementation.

1. OPERATING SYSTEM FUNCTIONS

Cold Start Operation

The operating system must be available immediately after power is applied to

the computer. In the past, some systems provided this capability by storing, in read-only memory, a small "bootstrap" routine. This bootstrap routine was then used to load the operating system into memory from another device, such as a paper-tape reader or a cassette recorder. New technology eliminates this tedious step. The operating system for your computer resides permanently on the EPROM (erasable-programmable read-only memory) chip and is ready to be executed as soon as power is applied and the "RESET" button is pressed. The depression and release of the "RESET" button sets the Z80 PC (program counter) to zero.

With the next machine cycle, the processor begins execution of the instruction located at 00₁₆ (location 00 hexadecimal) in memory. The operating system of the Z80 microprocessor provides the instructions to begin execution. This particular series of program instructions constitutes a "cold start" procedure and establishes the required start up conditions for the operating system. The operating system then initializes the SP (stack pointer) to an area in programmable memory for maintaining the "push-down/pop-up" stack. This stack is required for execution of any of the "RESTART" and "CALL" instructions provided by the Z80 instruction set. If it were not initialized before the execution of a "CALL" or "RESTART" instruction, the effects of the instruction would be unpredictable. In this operating system, the stack pointer is set to programmable memory location 07C4₁₆.

Warm Start Operation

After initializing the SP address, the operating system enters a command recognition module. Before discussing this feature of the operating system, some of the other restart features should be explained. The Z80 gives the user eight address-vectorized "RESTART" instructions (see Chapter 3 for a description of the instructions). For example, the execution of a RST 08₁₆ will store the current PC on the "STACK" and program execution will begin at location 08₁₆.

The following "RESTART" instructions are available within the operating system:

RST	10 ₁₆
RST	18 ₁₆
RST	20 ₁₆
RST	28 ₁₆
RST	30 ₁₆
RST	38 ₁₆

The execution of any of these instructions causes the operating system to jump to a location in programmable memory. At that location the user executes a jump instruction to vector the computer to a new location.

RST 00₁₆ and RST 08₁₆ have been reserved for use by the operating system for special functions and will not result in a jump to a location in programmable memory. These two RST instructions can be utilized in the debugging of programs. RST 00₁₆ will perform the same function as pressing the "RESET" button; or it will reinitialize the stack pointer and enter the command recognition module through execution of the "cold start" routine.

The execution of a RST 08₁₆ by the Z80 will result in the "warm start" module being entered. This module saves the existing data in all the registers in the "register save area" located in programmable memory (see the listing of the ZAP operating system in Appendix D). The module will also extract from the stack the user's restart address and save this in the register save area. The operating system then enters the command recognition mode to wait for the next command. The use of this feature allows the programmer to save register, pointer, flag, and program counter data, prior to using any additional debugging features in the operating system. A detailed description of the "warm start" module is provided in section II.2 of this chapter.

Program Development and Debugging Services

The cold start and warm start procedures exit to the command input sequence. With these command procedures, the programmer is able to examine and replace data in memory or registers, and to begin execution at a user-specified location. Upon entry to the command input module, the operating system displays "FFFF" on the address section, and "FF" on the data section of the six character hexadecimal LED display. The user then implements one of the three command functions by holding down the "SHIFT" key and pressing the "0", "1", or "2" keys. A "SHIFT 0" (the SHIFT key and 0 key are pressed simultaneously) tells the operating system to enter the memory display and replace function; "SHIFT 1" enters the register display and replace function, and a "SHIFT 2" enters the go execute module.

Memory Display and Replace

The memory display and replace function allows the user to examine the contents of both read-only memory and programmable memory. During operation the address and the contents of that location are shown on the respective displays.

The memory display and replace function is entered by executing a "SHIFT 0" when the system is in the command recognition mode (address display = FFFF and data display = FF). At this time, the operating system is waiting for the user to enter an address of one to four hexadecimal digits from the keyboard. As entered, these shift into the display area sequentially. If more than four digits are entered, only the last 4-digit value (shown in the address display) will be used as the address. Inputting of address data is terminated by pressing the "NEXT" key. This causes the contents of the indicated address to be displayed on the two digit hexadecimal data display. If the user wishes to display subsequent memory locations, he need only continue pressing the "NEXT" key. This will step the memory display program to the next higher memory location and display the new address and memory contents. If the user wishes to change the contents of a displayed memory location, he may enter new data by typing a two-digit value for that location before hitting the next key. This new value is loaded into the indicated address when the "NEXT" key is pressed. Pressing the "NEXT" key continues the sequential display of address and data.

Termination of this function is accomplished by pressing the "RESET" or "EXEC" buttons. Control is returned to the command recognition portion of the operating system.

Display Memory Example

Key	Address Display	Data Display
	FFFF	FF
"SHIFT 0"	0000	FF
1	0001	FF
A	001A	FF
F	01AF	FF
"NEXT"	01AF	01
"NEXT"	01B0	1C
"RESET"	FFFF	FF

Memory Replace Example

Key	Address Display	Data Display
	FFFF	FF
"SHIFT 0"	0000	FF
4	0004	FF
0	0040	FF
0	0400	FF

"NEXT"	0400	01
2	0400	02
1	0400	21
"NEXT"	0401	05
6	0401	06
A	0401	6A
"EXEC"		
The results will be:	Address	Data
	0400	21
	0401	6A

Register Display and Replace

The register display and replace function allows the user to examine and change the contents of the saved Z80 registers. This is accomplished by executing a RST 1 (warm start) during the execution of the program. During execution of this function, the contents of the registers are shown on the address display. Eight-bit registers will be displayed on the lower two digits of the address display. (The upper two digits will be zeros during the display of 8-bit registers.) A code that indicates which register is being displayed is shown on the data display. Table 6.1 describes the codes that have been assigned to the register display and replace function, as well as the key that initiates a particular register display sequence.

Code (shown on data display)	Z80 Register (shown on address display)	Initiating Key
02	IX	2
03	IY	3
04	SP	4
05	PC	5
06	I	6
07	R	7
08	L	8
09	H	9
0A	A	A
0B	B	B
0C	C	C
0D	D	D
0E	E	E
0F	F	F
40	L'	"SHIFT 0"
41	H'	"SHIFT 1"
42	A'	"SHIFT 2"
43	B'	"SHIFT 3"
44	C'	"SHIFT 4"
45	D'	"SHIFT 5"
46	E'	"SHIFT 6"
47	F'	"SHIFT 7"

Table 6.1 Display code/Z80 register/initiating key correspondence.

The register display and replace function is entered by pressing a "SHIFT 1" when the system is in the command recognition mode (address display = FFFF and data display = FF). At this time the operating system is waiting for the programmer to enter the one-digit register code (see table 6.1). If more than one digit is entered, only the last code indicated on the data display will be used as the reg-

ister identifier. When the central processor detects that the "NEXT" key has been depressed, the contents of the indicated register are displayed on the address display.

If the user wishes to display subsequent registers he need only press the "NEXT" key. This causes the next register to come up with the register code and its contents. To change the contents of a displayed register the value is entered and loaded when the "NEXT" key is pressed. For 16-bit registers, the last four hexadecimal digits will be accepted if more than four characters have been entered. For 8-bit registers the last two hexadecimal digits will be accepted. When replacing register data, the "NEXT" key also causes the register code to be indexed to the next register (see table 6.1) and its contents to be displayed.

The user may terminate this function by pressing the "EXEC" key. Control is returned to the command recognition portion of the operating system.

Display Register Example

Key	Data Display (register code)	Address Display (register contents)
	FF	FFFF
"SHIFT 1"	00	FFFF
A	0A	FFFF
"NEXT"	0A	005C
"NEXT"	0B	0063
"RESET"	FF	FFFF

Register Replace Example

Key	Data Display (register code)	Address Display (register contents)
	FF	FFFF
"SHIFT 1"	00	FFFF
5	05	FFFF
"NEXT"	05	043A
4	05	0004
2	05	0042
C	05	042C
"NEXT"	06	00FF
"NEXT"	07	0003
"EXEC"		

Go Execute ("EXEC")

The "go execute" ("EXEC") function allows the user to change the contents of the PC (program counter) register in order to direct execution of instructions at the user-selected address.

The "go execute" function is entered by pressing a "SHIFT 2" when the system is in the command recognition mode. Now the user must enter an address of one to four hexadecimal digits. If more than four digits are entered, only the value shown in the address display is used as the address to begin program execution. Execution begins when the "NEXT" or "EXEC" keys are pressed. This causes the Z80 registers to be stored in the register save area (see the operating system listing in Appendix D) and execution begins at the user-specified address.

GO EXECUTE Example

Key	Address Display	Data Display
	FFFF	FF
"SHIFT 2"	0000	FF
1	0001	FF
A	001A	FF
C	01AC	FF
F	1ACF	FF
"NEXT" or "EXEC"		

Serial I/O Services

The ZAP computer includes a serial input/output capability that is implemented with a UART. This interface allows serial communication between the computer and peripheral devices such as a printer or a CRT. To aid the user in utilizing this capability, the operating system has a UART diagnostic module, a serial input module, and a serial output module. The input and output modules are set up as subroutines that can be called during program execution and that are not necessarily keyboard and display limited.

UART Diagnostic Module

The UART diagnostic module provides a means for checking the performance of the UART. To utilize this feature the user must first attach the serial output and input lines together so that data output from the UART may be read by the same device. The serial diagnostic subroutine is initiated by using the "go execute" function. Execution starts at 032D₁₆.

Once started, the diagnostic module (UATST) begins by sending data to the UART and waiting for data to become available. The status of the UART is checked to verify that no fault conditions are present. In the event that a fault is detected, the status of the UART is displayed on the two low-order digits of the address display. (See table 6.2 for error codes.) If there are no errors, the data is read and displayed on the two-digit-data display. A comparison is made between the input and output data. If the 2 bytes are equal, the output character is incremented and another byte is sent to the UART to continue the sequence. This procedure continues until the "RESET" button is pressed, or until an error is detected. In the event that the input character does not equal the output character, a 0F₁₆ is displayed in the two lower digits of the address display and the diagnostic is halted. Figure 6.1 details the logic flow of this software routine.

Displayed Code	Error
12 ₁₆ or 13 ₁₆	Parity Error
0A ₁₆ or 0B ₁₆	Framing Error
06 ₁₆ or 07 ₁₆	Overrun Error
00	Transmitter Buffer Not Empty
0F ₁₆	Input Character ≠ Output Character

Table 6.2 UART error codes.

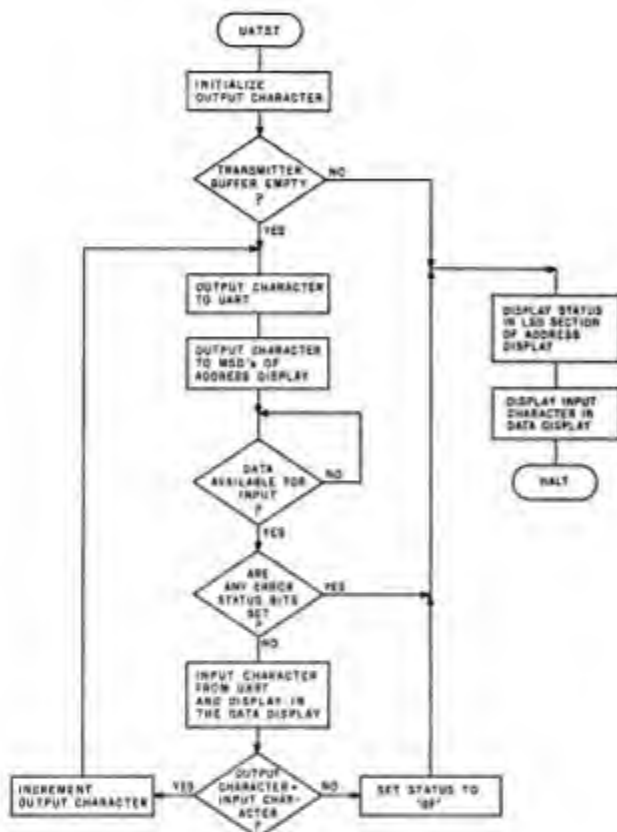


Figure 6.1 A flowchart of the UART diagnostic module (UATST).

Serial Input Module

The serial input module has been included so the user can read serial data from external devices. To utilize this capability, the user must set aside a programmable memory buffer where the input data is to be stored, and designate the number of input characters expected. The input buffer address is stored at address 07F9₁₆ in memory (see Appendix D), and the number of characters is stored at address 07FD₁₆. The communication reception begins when the TTYINP module is called.

Serial Input Initiation Example

TTYINP	EQU	035F ₁₆	Address of input module
BUFFER	EQU	07F9 ₁₆	Input buffer address
NCHAR	EQU	80	Number of characters to be received
TTYIBU	EQU	07F9 ₁₆	Operating system address constant
TTYIC	EQU	07FD ₁₆	Operating system address constant
	LD	HL, BUFFER	Set buffer for operating system
	LD	(TTYIBU), HL	

LD A, NCHAR	Set character count for operating system
LD (TTYIC), A	
CALL TTYINP	Call UART serial input routine

The data read by the serial input module will be stored in the user-specified buffer until the input sequence is terminated. When this occurs, control is returned to the user's program at the next instruction. Termination of the input process may be due to any of the following conditions:

- A status error is detected
- The number of characters read equals preset count
- The receipt of a carriage return as an input character (ASCII 0D_{ah})

In the event that a status error is detected, the A register will be equal to 80_{ah} when control is returned to the user. If termination results from filling the character buffer correctly, the A register will be equal to 00_{ah}. However, if termination is the result of a carriage return, the A register will be equal to the number of characters remaining to be input. Figure 6.2 details the logic flow of the TTYINP software module.

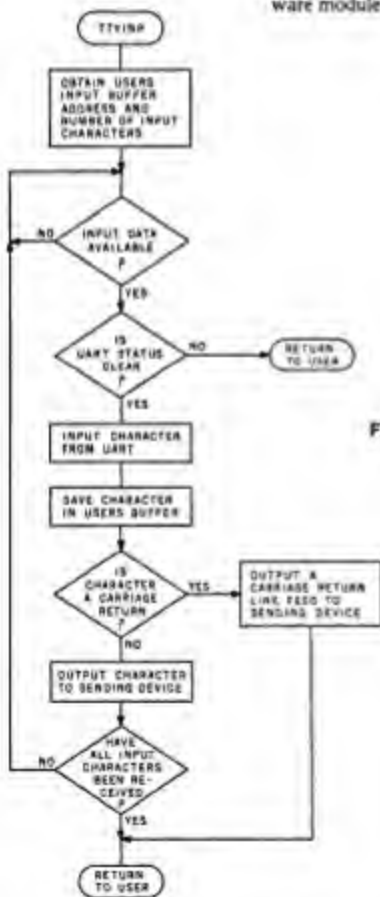


Figure 6.2 A serial input module (TTYINP) flowchart.

Serial Output Module

The serial output module is provided to assist the user in communicating serial output data to external devices. To use this module, the operator designates an output data buffer address and the the number of characters (bytes) to be transmitted. The output buffer address must be stored at 07FB₁₆ in memory (see Appendix D) and the number of characters to be sent is stored at address 07FE₁₆. Data transmission starts when TTYOUT is called.

Serial Output Initiation Example

TTYOUT	EQU	039E ₁₆	Address of output module
BUFFER	EQU	07FB ₁₆	Output buffer address
NCHAR	EQU	35	Number of characters to be transmitted
TTYOBF	EQU	07FB ₁₆	Operating system address constant
TTYOC	EQU	07FE ₁₆	Operating system address constant
	LD	HL, BUFFER	Set buffer address for operating system
	LD	(TTYOBF), HL	
	LD	A, NCHAR	Set character count for operating system
	CALL	TTYOUT	Call UART serial output routine

Control will be returned to the user when

- The output buffer is empty
- The transmit buffer does not become available, indicating an error

In the event that a normal termination occurs, the A register will be equal to 00₁₆ when control is returned to the user. However, if a premature termination and return are required, the A register will be equal to 01₁₆. Figure 6.3 details the logic flow of the serial output software module.

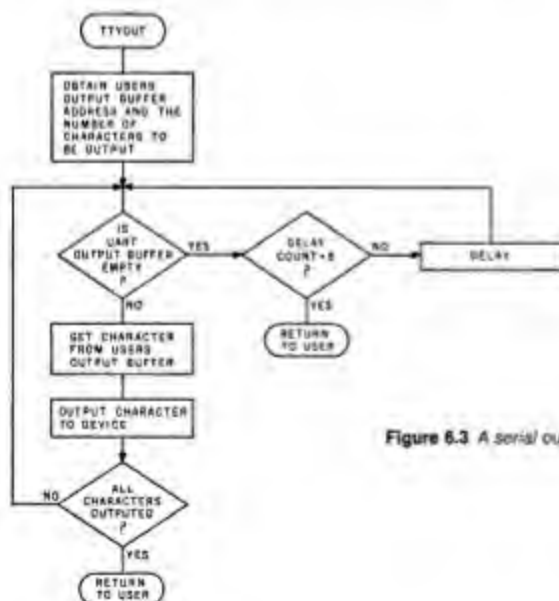


Figure 6.3 A serial output module (TTYOUT) flowchart.

II. Operating System Module Description

II.1 Warm Start Module

The warm start module (WARM1) is responsible for saving all Z80 registers in the register save area allocated in the reserved portion of programmable memory (see Appendix D). Upon entry, the user's A, H, and L registers are saved to provide working registers for the remainder of the module operation. Next, the user's PC is removed from the stack and is saved in the memory locations reserved for it.

The AF register pair is pushed onto the stack and popped off into the HL register pair. This procedure enables the flag register to be saved in the register save area. The remainder of the user's working and alternate registers are examined and transferred to the register save area. Upon completion of this task, the module exits to the command recognition module. (See Appendix D for additional details.) Figure 6.4 details the logic flow of the warm start module.



Figure 6.4 A flowchart of the warm start module (WARM1).

II.2 Command Recognition Module

The command recognition module (WARM2) is entered after the completion of a cold or warm start sequence. When initiated, the module clears the keyboard input buffer and the keyboard flags. This removes ambiguity for future operations. The module will set the data display to FF and the address display to FFFF. When completed, the module enters the KEYIN subroutine to get an input character from the keyboard. Any input character is checked to see if it corresponds to one of the three allowable functions. If so, control is transferred to the proper function; otherwise, the input is ignored and the module waits for the next input from the keyboard. (See Appendix D for additional details.) Figure 6.5 illustrates the logic flow of the command recognition module.

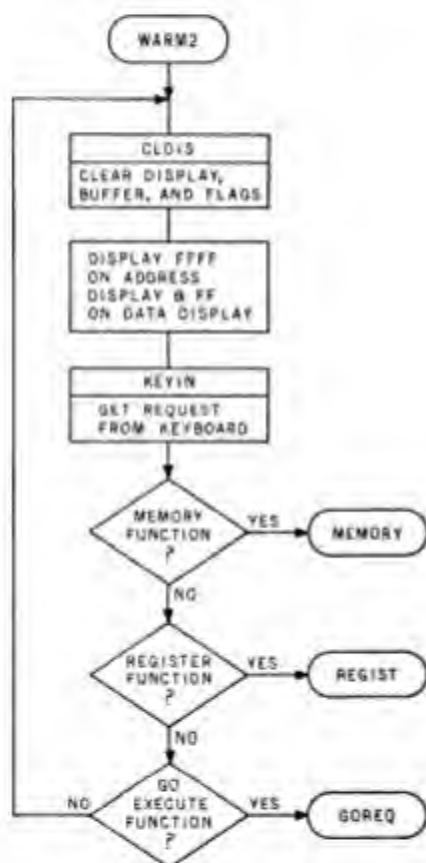


Figure 6.5 A flowchart of the command recognition module (WARM2).

II.3 Restart Module

The restart module (RESTRT) takes the values stored in the programmable memory register save area. It then restores the user's 8- and 16-bit registers before returning control to the location specified in the PC save area. This procedure restores the alternate registers, and then the working registers. In either instance, the flag registers are restored by pushing the data onto the stack and then popping it off to the F register. In order to exit to the user's restart address, the saved PC is pushed onto the stack and a "RET" (return instruction) is executed. (See Appendix D for additional details.) Figure 6.6 details the logic flow of the restart module.



Figure 6.6 A flowchart of the restart module (RESTRT).

II.4 Keyboard Input Module

The keyboard input module (KEYIN) provides the primary interface between the computer and the user. Upon entry, it begins to read data from the keyboard input port. It stays in a loop, checking the MSB (most significant bit) of the data. The MSB is the key-pressed strobe. When it goes to a logic one level, the seven LSBs (least significant bits) of the keyboard input port are retained as the desired input character. The module then returns to the user's program with the keyboard character in the accumulator. (See Appendix D for additional details.) Figure 6.7 details the logic flow of the keyboard input module.

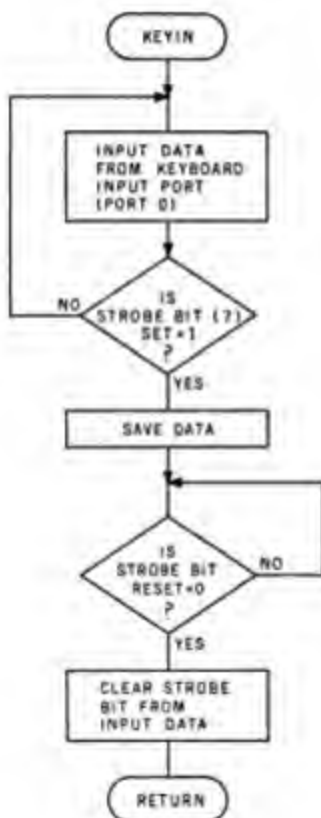


Figure 6.7 A flowchart of the keyboard input module (KEYIN).

II.5 One Character Input Module

The function of this module (ONECAR) is to input one or more characters from the keyboard. This module also indicates the last character and whether it was accompanied by a "NEXT" or "EXEC" key.

Upon entry, the input buffer and keyboard flags are cleared. (The data display may or may not be cleared depending on the requirements of the calling module.) The module waits for an input character to be passed to it. When it receives a character, it checks to see if it is a "NEXT" "EXEC", or valid data. In the event that the input is a "NEXT" or "EXEC", the appropriate keyboard flag is set along with the no data flag and control returned to the user (see figure 6.8).

If an invalid data character is received, the module is reinitiated. Upon receipt of valid data, the data is stored in a 1-byte input buffer, and the module waits for the next input character. This character is processed in a manner similar to the one just described with the following exception: in the event that the input character is a "NEXT" or "EXEC", only the appropriate flag is set before returning control to the user. (See Appendix D for additional details.) Figure 6.9 shows the logic flow of the one character input module.

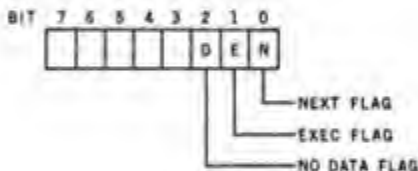


Figure 6.8 The configuration of the keyboard flags.

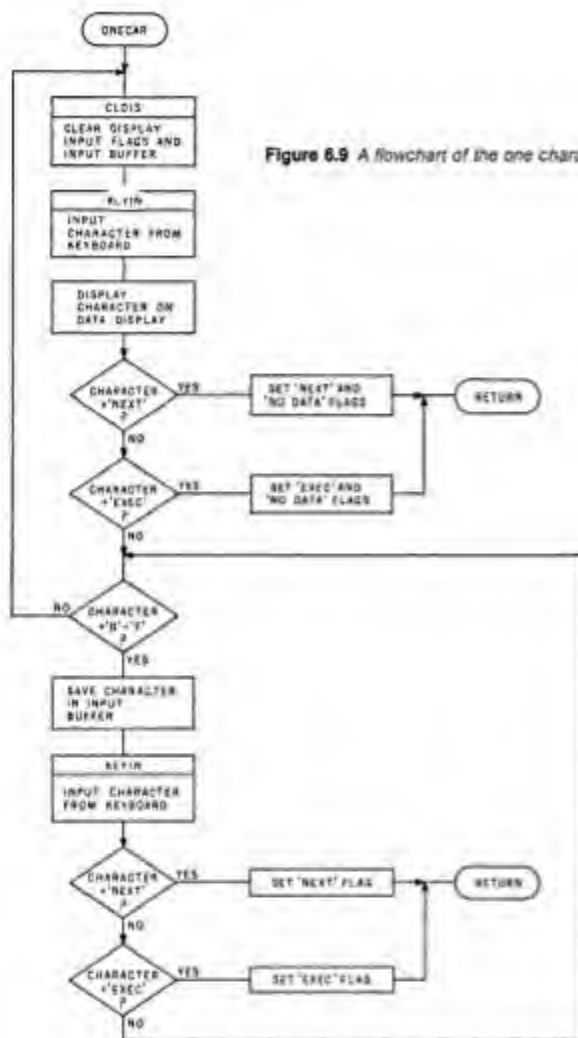


Figure 6.9 A flowchart of the one character input module (ONECAR)

II.6 Two Character Input Module

The function of this module (TWOCHAR) is to input one or more characters from the keyboard and transfer to the user the last two characters when a "NEXT" or "EXEC" key is pressed. The module also notifies the user of the type of termination that took place.

Upon entry, the input buffer and keyboard flags are cleared. (The data display may or may not be cleared depending on the requirements of the calling module.) This module calls the keyboard input module to obtain its input data. The first character is checked to determine if it is a "NEXT" or "EXEC"; the appropriate keyboard flag is set along with the no data flag, and control is returned to the user (see figure 6.8). If an invalid character is received, the module is reinitiated.

The receipt of valid data will cause the module to format the data as a two-digit value in the keyboard input buffer. It then returns to the user with the appropriate flags set. (See Appendix D for additional details.) Figure 6.10 details the logic flow of the two character input module.

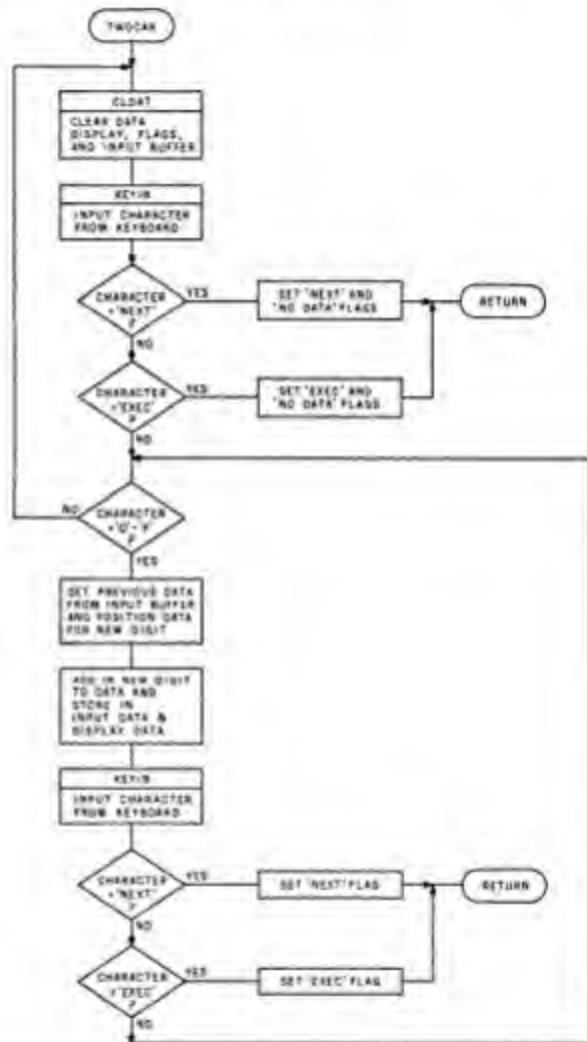


Figure 6.10 A flowchart of the two character input module (TWOCHAR).

II.7 Four Character Input Module

The function of this module (FORCAR) is to input one or more characters from the keyboard and to transfer to the user the last four characters when a "NEXT"

or "EXEC" key is pressed. In the event that less than four characters are input, the higher order digits will be set to zero. The module also notifies the user via the keyboard flags (see figure 6.8).

The operation of this module is very similar to the two character input module. The main difference lies in the manner in which the new data (input from the keyboard) is merged into previous input data from the keyboard. (See Appendix D for additional details.) Figure 6.11 shows the logic flow of the four character input module.

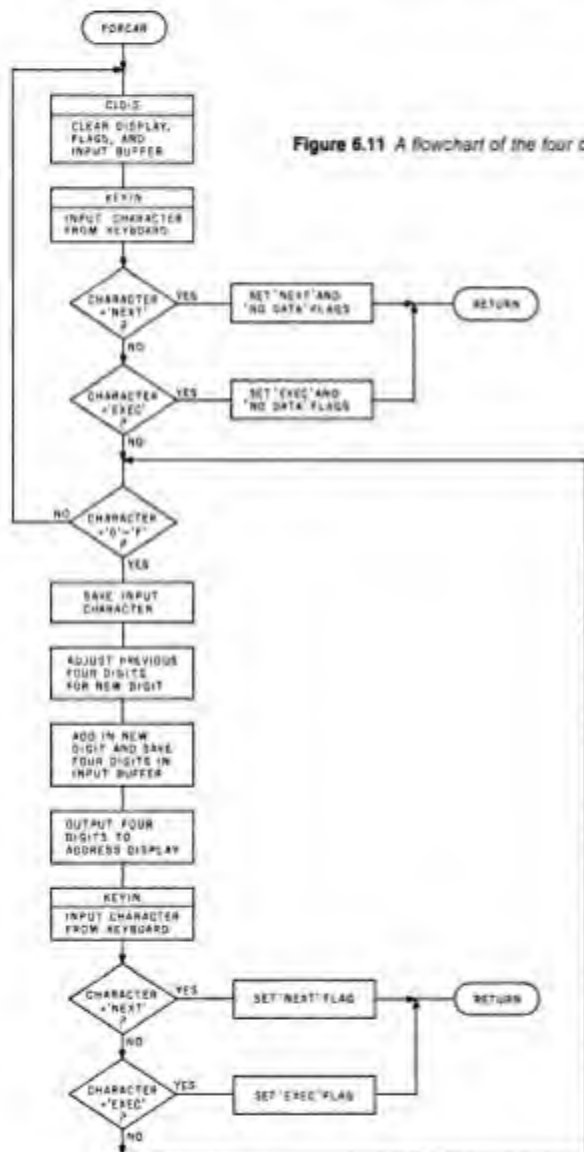


Figure 6.11 A flowchart of the four character input module (FORCAR).

IL8 Memory Display and Replace Module

The memory display and replace function is one of the three major modules of the operating system. Upon entry (see command recognition module), this module (MEMORY) makes a call to FORCAR (four character input module) to get the base memory address at which to begin displaying the memory contents. When it returns from FORCAR, the keyboard flags are examined to determine if the "EXEC" flag is set (=1). In the event that the "EXEC" flag is set, control is transferred to the restart module (RESTRT). If the "EXEC" flag is not set (=0), the address location and memory contents are output to the appropriate displays. The TWOCAR (two character input module) is called to obtain new data from the displayed memory location.

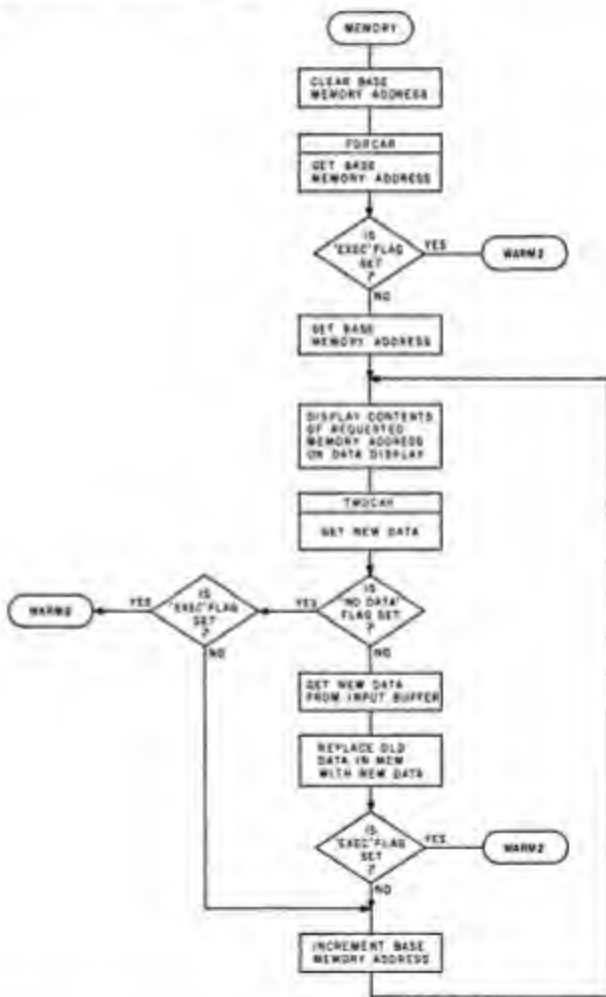


Figure 6.12 A flowchart of the memory display and replace module (MEMORY).

When control is returned from TWOCAR, the module checks the "no data" flag in the keyboard flag word. If this flag is set (=1), the "EXEC" flag is examined. If that is set, control is transferred to the command recognition module (WARM2). If, on the other hand, the "EXEC" flag is reset (=0), the user's memory address is incremented, displayed on the address display, and its contents are displayed on the data display.

If, on return from TWOCAR, the "no data" flag is reset (=0), the new data is extracted from the keyboard input buffer and stored in the displayed memory location. At this time, the module determines if TWOCAR was exited via an "EXEC" or "NEXT" directive. In the event that the "EXEC" flag is set (=1), control is transferred to the command recognition module (WARM2). If, however, the flag is reset (=0), the user's memory address is incremented, displayed on the address display, and its contents are displayed on the data display. Then the two character input module is called to get the next directive for the memory display and replace module. (See Appendix D for additional details.) Figure 6.12 shows the logic flow of the memory display and replace module.

11.9 Register Display and Replace Module

The register display and replace module (REGIST) is one of the three major modules of the operating system. This module calls the ONECAR (one character input module) to get the initial register display code from the user (see table 6.1). Upon return from ONECAR, the "EXEC" flag is checked. If this flag is set (=1), control is transferred to the command recognition module (WARM2). If the "EXEC" flag is reset (=0), the base register display index is calculated from the user's register display code.

At this time, the register index is checked to see if the register request is an 8- or 16-bit register. If the user requests a 16-bit register, the appropriate register code is displayed in the data display, and the requested register data is obtained from the register save area and displayed in the address display. The module then makes a call to the FORCAR (four character input module) to get new data for the register. Upon return, the "no data" flag is checked. If this flag is set and the "EXEC" flag is set, control is transferred to the RESTRT (restart module). If the "no data" and "NEXT" flags are set, the register display index is incremented and displayed in the data display. The new register data is obtained from the register save area and displayed on the address display.

If an 8-bit register has been requested, the register code (see table 6.1) is displayed in the data display, and the appropriate data is obtained from the register save area and displayed on the address display. At this time, the module calls TWOCAR to get new data from the displayed register. When the two character input module returns control, the module determines the mode of execution by examining the keyboard flags. If the "no data" and "EXEC" flags are set, control is transferred to the command recognition module (WARM2). If the "no data" and "NEXT" flags are set, the register index is incremented and the register contents channeled to the appropriate display.

If the "no data" flag is reset, the new register data is obtained from the keyboard input buffer and stored in the appropriate register save location. At this time the "EXEC" flag is checked and, if set, control is transferred to the command recognition module (WARM2). If the "EXEC" flag is reset, the register data is displayed and the user directive processed. (See Appendix D for additional details.) Figure 6.13 details the logic flow of the register display and replace module.

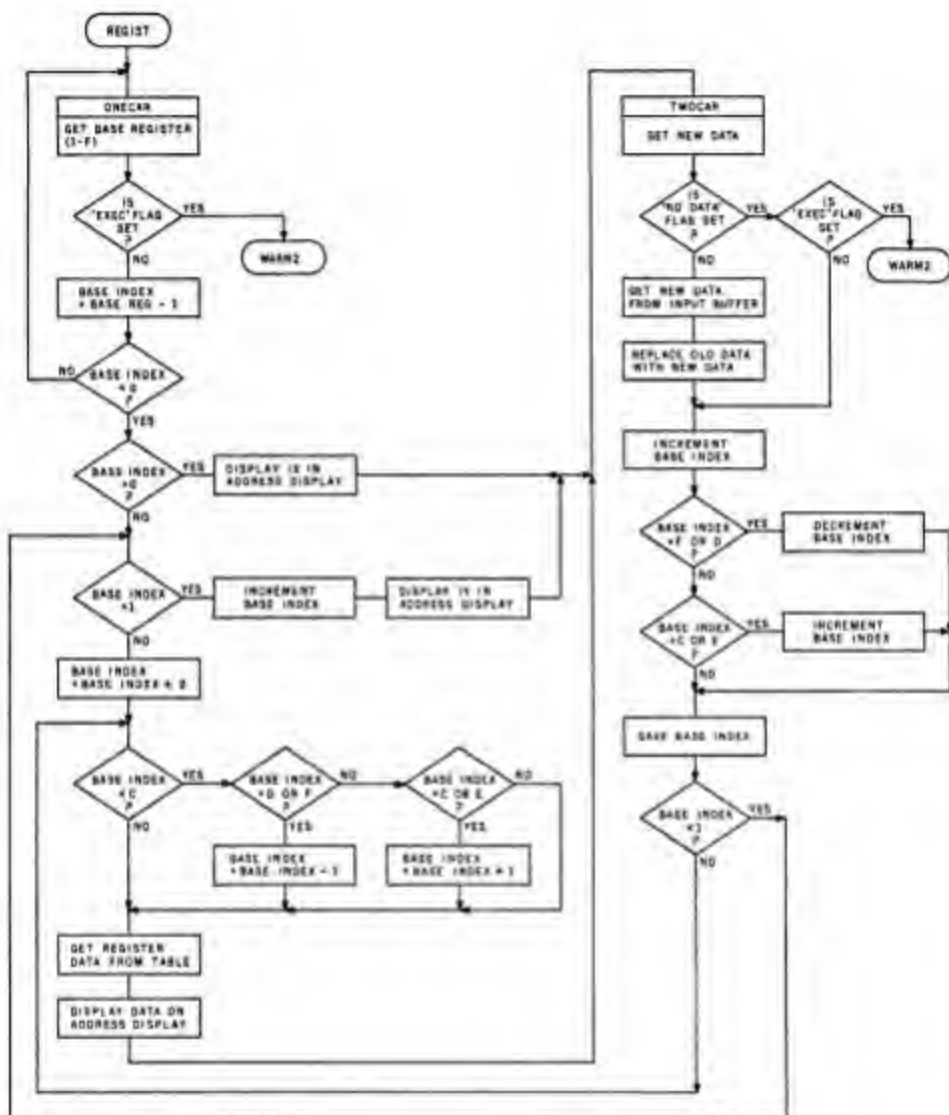


Figure 6.13 A flowchart of the register display and replace module (REGIST).

II.10 Go Execute Module

The go execute module (GOREQ) is the last of the three major functions of the operating system. Upon entry (see command recognition module), this module calls FORCAR to get the address where execution is to begin. Upon return from FORCAR, the "no data" flag is examined to determine the mode of execution. If this flag is set (=1), control is immediately transferred to RESTRT. This restores the Z80 registers and resumes execution at the PC address currently contained from the keyboard input buffer and stored in the PC save location in the register save area. Control is then transferred to the command recognition module (WARM2) which will restore the registers with the saved data, and begin execution of the user's program at the specified address. (See Appendix D for additional details.) Figure 6.14 details the logic flow of the go execute module.

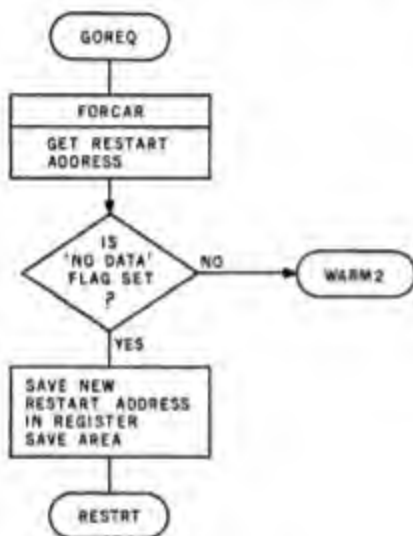


Figure 6.14 A flowchart of the go execute module (GOREQ).

CHAPTER 7

PROGRAMMING AN EPROM

The ZAP computer has been designed to be inexpensive, reliable, and easy to construct. To keep costs and complexity to a minimum, some computer features that could be helpful to a beginner have been eliminated. The most visible of the missing features are a front panel and display. While this in no way detracts from the operation of the computer, its inclusion would make initial checkout and program development easier.

To properly test ZAP, a program must be in memory. This program does not have to be very long—only a few instructions are necessary to determine whether the computer runs at all. The problem arises when the user wishes to run a program of 50 or 100 bytes in length. We end up with a “catch-22” situation. To effectively enter machine code into ZAP’s programmable memory, a program that coordinates this activity must be running in EPROM. Such a program is called a monitor and is outlined in Chapter 6. The catch is that writing the monitor software into an EPROM automatically requires the monitor to be running the programmer. Fortunately, if one has an alternate way of writing the 1 K ZAP monitor into EPROM, this is no longer a problem.

Rather than leaving the experimenter to his own devices, this section includes information on programming EPROMs. To solve the startup situation, I’ve outlined a design for a couple of manual EPROM programmers. Loading programs on a manual programmer is tedious. They are primarily intended for much shorter routines such as checking basic system operations. However, one manual unit can be modified to load the full 1 K monitor software. When ZAP is fully operational, you can use it in conjunction with an automatic programmer. This will help in writing a number of EPROMs. In the event that you do not wish to write your own EPROM, consult Appendix A for the availability of preprogrammed EPROMs.

A Quick Review of EPROMs

It is often desirable to have the non-volatility of ROMs but the read/write capabilities of semiconductor programmable memories. An effective compromise is the EPROM. This is a read-mostly memory. It is used as a ROM for extended periods of time, occasionally erased and reprogrammed as necessary. Erasure is accomplished by exposing the chip substrate, covered by a transparent quartz window, to ultraviolet light. We’ll cover erasure at the end of this chapter.

The EPROM memory element used by Intel and most other manufacturers is a stored charge type called a FAMOS transistor (Floating-gate Avalanche injection Metal Oxide Semiconductor) storage device. By selectively applying a 25 V charging voltage to addressed cells, particular bit patterns that constitute the program can be written into the EPROM. This charge, because it is surrounded by insulating material, can last for years. Exposure to intense ultraviolet light drains the charge and results in the erasure of all programmed information.

There are many EPROMs on the market—2708s, 2716s, and 2732s are the major ones. For the most part, computerists have moved away from the very difficult-to-program 1702s and have opted for the more easily programmed 2708s and 2716s. An added benefit is their greater storage density. The newer EPROMs on the market are considerably more expensive than the 2708. All things considered, the 2708 is the best

buy for the money. At slightly greater expense, you could use the 2758 for a single supply operation. For these reasons, the EPROM programmer outlined in this chapter is the 2708.

Figure 7.1 is the circuit for a manual 2708 programmer. IC 5 and two sections of IC 3 provide the +25 V program pulse to the EPROM. IC 5 is set for a duration of 1 ms and is triggered by a logic 0 to 1 transition at its input. The EPROM both sources and sinks current through programming pin 18. A combination of devices rather than a simple open-collector driver is necessary. In the write mode, when \overline{CS}/WE pin 20 is at +12 V and between programming pulses, pin 18 has to be pulled down by an active device because it sources a small amount of current. The programming pulse itself is about 30 mA and cannot easily be accommodated without emitter-follower configured Q1. This pulse should be between 25 and 27 V at pin 18. Three 9 V batteries will suffice. (An alternative is to use a commercial encapsulated 24 V, 50 mA power supply. The encapsulated supply can be resistor trimmed to produce the desired 25 to 27 V.)

To write a byte into the EPROM, a 10-bit address designating which of the 1024 bytes will receive the data is preset on switches SW 1 thru SW 10. To start at location 0, all switches will be in the closed position. Next, the 8 bits that are to be stored are set on switches SW 12 thru SW 19. This data byte should be reflected on the output display LED 1 thru LED 8. Finally, to get the programmer in the write mode, switch SW 11 is set open. Actual insertion of the data occurs when the write pulse pushbutton PB 1 is pressed. This fires a 1 ms pulse of 25 V into the 2708 program pin. According to manufacturer's specifications, no single programming pulse should be longer than 1 ms. For maximum data retention, 100 of these programming pulses are recommended (totalling 100 ms per byte).

Unfortunately, 100 ms cannot be applied to a single address all at once. Manufacturers specify that it should be done sequentially and should consist of 100 1-ms applications. In short, it means that for a 25-byte program, each address should be written with one pulse and then the loop repeated up to 100 times. I have never tried to lengthen the pulse and program a 2708 faster than called for. Experience has shown, however, that some EPROMs are completely written with as few as 2 or 3 loops. Obviously, for full retention each address should be rewritten on an automatic programmer.

Reading back the stored contents of a 2708 is easy on the same manual programmer. First, all data input switches SW 12 thru SW 19 are opened to the "1" state and then "read/write" switch SW 11 is set in the closed or "read" mode. No other pulsing or clocking is necessary. The output display will show the contents of the byte pointed to by the address input switches SW 1 thru SW 10. It will remain constant until set to another address. Reading out the contents is simply a matter of incrementing this 10-bit address through the range of program addresses.

A slightly more complex manual programmer is demonstrated in figure 7.2. Three presettable counters are inserted between the address input switches and the EPROM. Instead of changing the switch positions for each address, they are now used only to preset the counters to some beginning address. If we want to program an EPROM starting at hexadecimal 3AA, the switches would be set to that address and the "address preset" switch pressed. The 10 LEDs, LED A0 thru LED A9, would read 3AA as the address. The data to be programmed is set on SW 12 thru SW 19. Pressing the "write data" push button PB1 (the renamed "address increment") stores the data from the switches. Successive memory locations are programmed by setting SW 12 thru SW 19 and pressing PB1. Resetting the address counter to zero is accomplished by pressing the clear button.

It is easy to see how this manual programmer, while not greatly improving programming time, facilitates reading memory. Put all the data input switches to the logic 1 level, set the interface to the read mode, and preset and load a start address. Readout is accomplished simply by repeated operation of the address increment button.

An Automatic Programmer

You will need an operational ZAP computer to build an automatic programmer. The

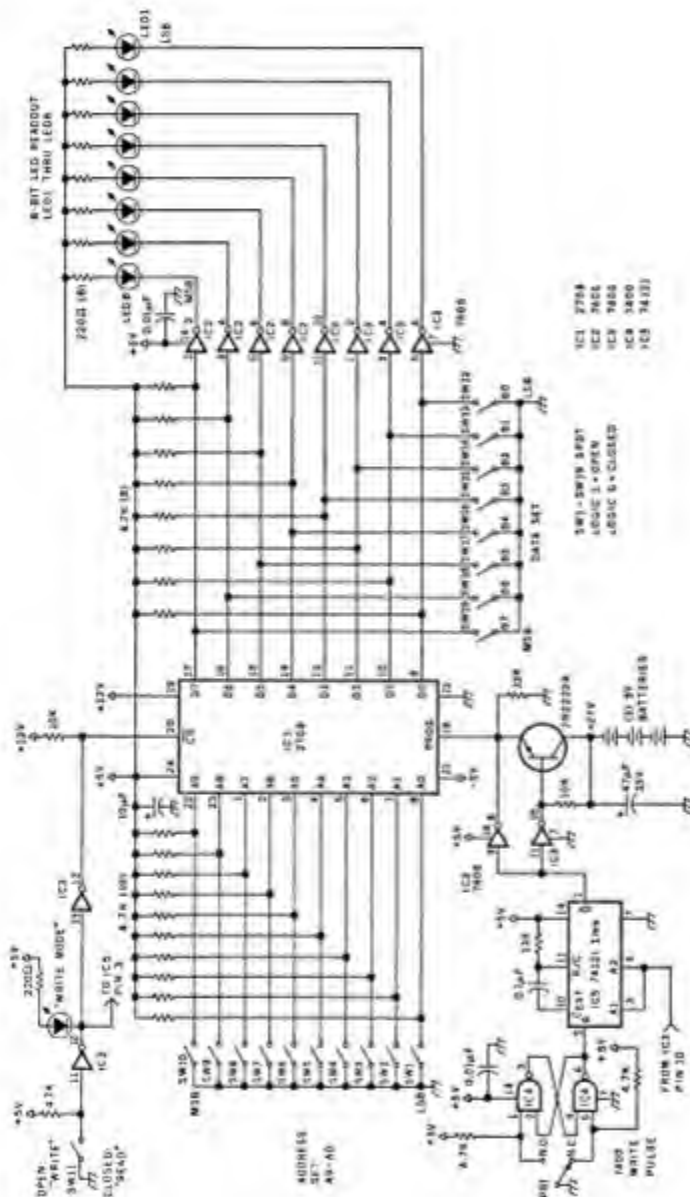


Figure 7.1 A schematic diagram of a manual 2708 programmer.

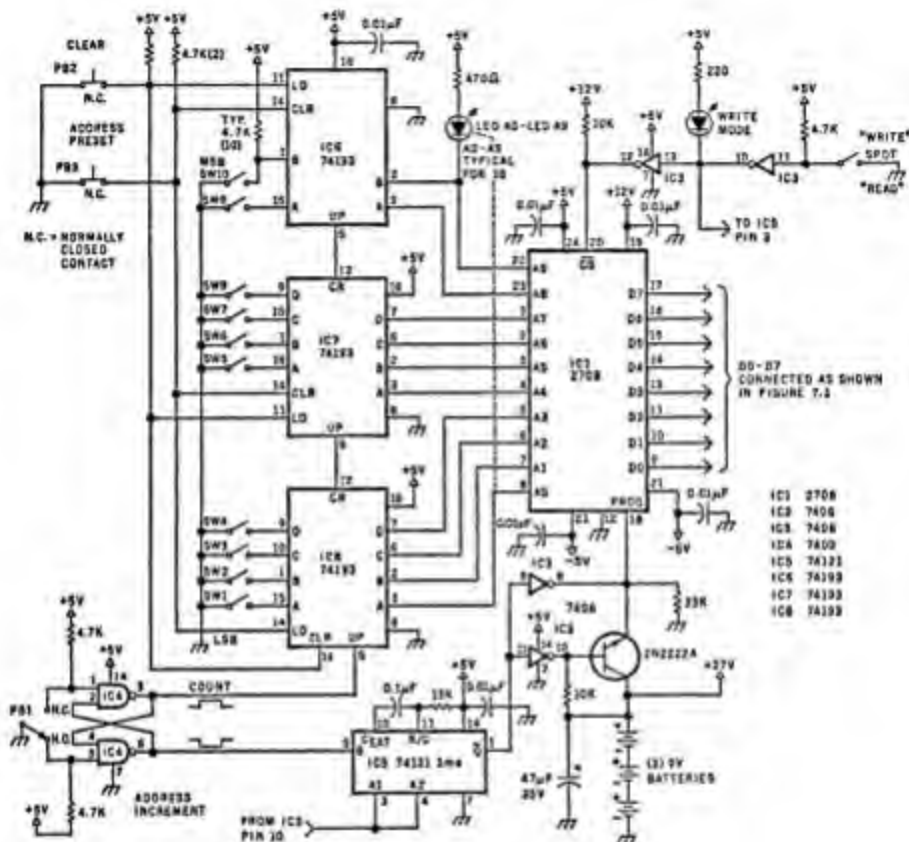


Figure 7.2 A schematic diagram of a self-incrementing manual 2708 programmer. Light-emitting diodes (LEDs) are to be connected to all 10 address-input lines of the 2708. For clarity, only one LED (connected to address line A9) is shown in the diagram. The other LEDs are to be wired in the same way.

complexity of design can be reduced considerably by taking advantage of decoded, but to this point unused, I/O strobes provided in the basic ZAP. The circuit shown in figure 7.3 takes three less chips than the manual programmer in figure 7.2. Its operations, while similar in operation, are quite different in detail.

Four I/O strobes (input and output port 1, and input and output port 4) synchronize the hardware and software. Figure 7.4 shows the logic flow for writing an EPROM. With the EPROM connected directly to the data bus, only the strobes, rather than full-latched registers, are necessary for this interface.

To write data, the sequence should be as follows: first, an OUT 04 pulses the address counter clear lines, setting them to 0. Next, the EPROM is set to the program mode, and the first byte is written into the EPROM with an OUT 01 instruction.

Figure 7.5 shows how the 2708 program mode is selected. The significance of this circuit is that its output is wired as a 2-bit digital-to-analog converter to control the chip-select line of the 2708.

When an OUT 04 is executed, the \overline{CS} pin will see 0 volts enabling the read mode. When an OUT 01 is executed, this voltage will be 12 V for program mode. When no strobe is present, \overline{CS} will be at +5 V and the 2708 will be three-state.

An OUT 01 fires the 25 V program pulse for 1 ms while the pertinent data is on the data bus. After that, an INP 01 is executed, which increments the address counter to the next address position. We are not actually doing any input function, but instead we are using the decoded strobe of the INP 01 instruction to mean "increment address register."

The hardware automatically keeps track of the address, but the software must implement its own counters to keep track of the 0 to 1023 positions as well as the number of times the complete 1024 bytes have been programmed. Remember, the manufacturer suggests 100 1-ms loops.

Reading the EPROM automatically is also very simple. A flow diagram of the logic is shown in figure 7.6. The address counter is cleared again by doing an OUT 04. Data is read by executing an INP 04. This data can be stored and analyzed. Finally, the address counter is incremented again with an INP 01, and the process is repeated to read the next byte.

While discussion has centered on the Intel 2708 EPROM as the most cost-effective choice, there are many other EPROMs on the market. Two devices of particular importance (should their price and availability improve by the time you read this) are the Intel 2758 and 2716. These are 1 K and 2 K single supply (+5 V) EPROMs, respectively. The significance for the experimenter is that these parts can be programmed with a single, 50 ms, 25 V program pulse to each address rather than successive 1-ms loops. The three programmer circuits presented are set up for 2708s but can be easily reconfigured for these other devices. Changing the one-shot timing pulse from 1 ms to 50 ms and rewiring a few pins will allow complete programming with just a single run through the addresses (they don't have to be successively programmed, either).

Erasing An EPROM

EPROMs bought directly from a manufacturer come completely erased. If you plan on writing an EPROM program once, and you either don't want to modify it or you don't make mistakes, forget about erasing. The majority of computerists will want to reprogram EPROMs. It then becomes necessary to know how to erase them. We all know that EPROMs are ultraviolet erasable. However, duration, distance from the light source, and intensity determine the quality of the erasure.

People concerned about maintaining a manufacturer's specifications during the programming sequence should also be advised of the proper erasing methods. Unlike the test read-after-write-loop method for programming, EPROMs are usually removed from the circuit during erasing. Therefore, it is advisable to perform the procedure correctly, or it will have to be repeated.

The typical 2708 EPROM can be erased by exposure to high intensity shortwave ultraviolet light, with a wave length of 2537 Å. The recommended integrated dose (UV intensity \times exposure time) is 12.5 watt-seconds per square centimeter (Ws/cm²). The time required to produce this exposure is a function of the ultraviolet light intensity.

Cost and safety, equally emphasized, should be the guiding factors when selecting an ultraviolet eraser. A commercial unit not only specifies its intensity (that allows computation of exposure time), but also includes important interlocks. It is conceivable that some homebrew erasers might have improper shielding that could allow the ultraviolet light to escape or be accidentally turned on while being viewed. Such possibilities can lead to permanent eye damage.

One of the more cost-effective erasers on the market is the UVS-11E by Ultra-Violet Products, Inc., San Gabriel CA, 91776. This unit is made especially for the home computer market and includes some important safety features. The lamp will not operate unless properly seated, and if lifted from its holding tray, it will automatically shut off. At the standard exposure distance of 1 inch, the UVS-11E produces an intensity of 5,000 μW per square centimeter ($\mu\text{W}/\text{cm}^2$). Exposure time for the 2708 is easily calculated.

Exposure time (T_E)

$$T_E = J + I$$

Where

J = required erasure density of device

I = incident power density of eraser

For a 2708 which requires 12.5 Ws/cm^2

$$J = 5000 \mu\text{W}/\text{cm}^2$$

$$I = 12.5 \text{ Ws}/\text{cm}^2$$

$$T_E = \frac{12.5}{5000 \times 10^{-6}} = 2500 \text{ seconds}$$

or $T_E = 41.6$ minutes for complete erasure

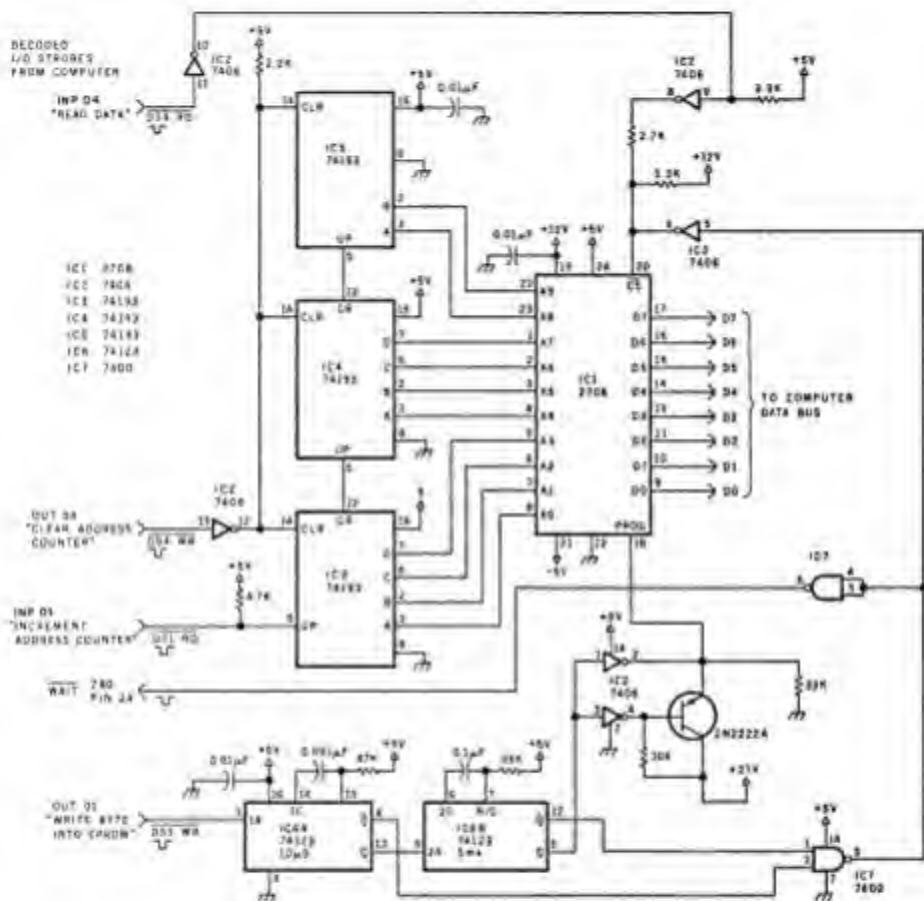


Figure 7.3 A schematic diagram of an automatic 2708 programmer.

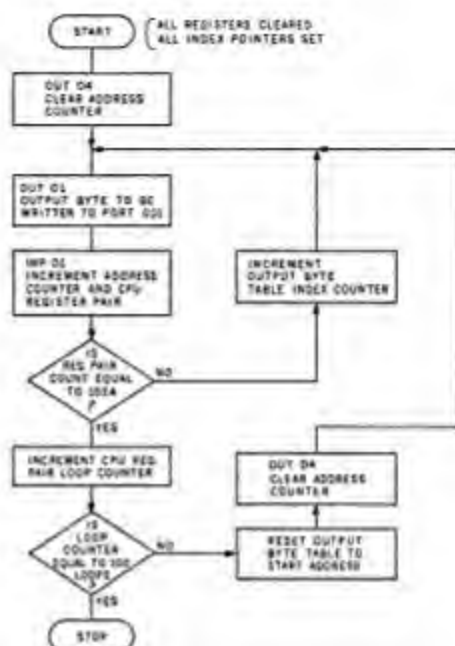


Figure 7.4 A flowchart of an automatic EPROM programmer write cycle.

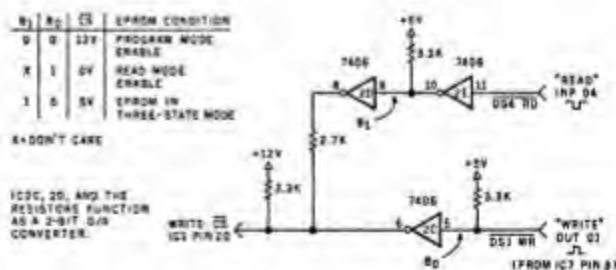


Figure 7.5 Programmable control of an EPROM \overline{CS} line in an automatic EPROM programmer.

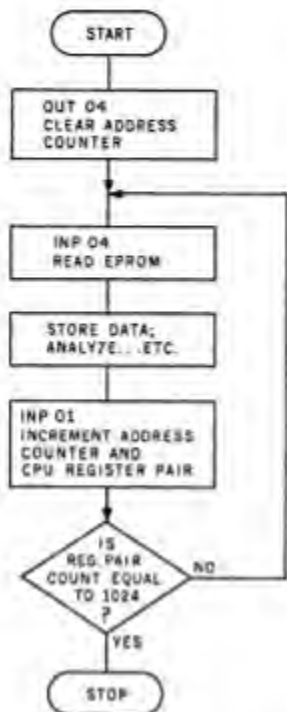


Figure 7.6 A flowchart of an automatic EPROM programmer read cycle.

CHAPTER 8

CONNECTING ZAP TO THE REAL WORLD

It's now obvious that the ZAP computer can be configured in a number of ways. Depending on your needs, you can go far beyond the basic system I have outlined. If you want a personal computer that is the equivalent of large commercial microcomputer systems, then you must add considerably more memory and peripherals. Accommodations must be made for a more powerful operating system and most probably a high-level language such as BASIC or Pascal. If you intend to use the ZAP computer as a word processing system, then a video display and printer will be required. This, in turn, necessitates adding more parallel and serial ports. Whatever the eventual configuration, the design considerations that went into constructing the ZAP computer do not change.

The ZAP computer is intended as a trainer. This book is structured in such a way that you should be able to lay out a system configuration and build it. I have not discussed what it takes to design a word processing system, or to add floppy disk storage, because it is beyond the scope of this introductory text. The support material necessary to adequately cover such an undertaking would be enough for another book. This does not mean, however, that everything is finished once the ZAP computer is constructed and you learn how to write and execute a short program. Quite the contrary; a more significant application of ZAP is to connect it to something considered part of the "real world" and have it perform some constructive task. ZAP's "power to weight" ratio makes it a natural for intelligent control applications. The real key to using ZAP effectively is learning how to connect it to the real world.

Within the framework of the direct examples I have outlined, the ZAP computer created from this book should be a single-board computer suitable for use in a variety of applications. Because it includes a serial port, two parallel ports, PROM monitor, and programmable memory, ZAP is in many respects equivalent to commercial digital controllers costing hundreds of dollars more.

Small single-board computers are most often used in data acquisition and intelligent control applications. Their function is usually to digest certain input parameters and compute a result. For example, in a 100 HP electric motor control, the inputs would be voltage, current and RPM, and the control output would be a load factor correction voltage.

In all probability, a few of these "intelligent controllers" were used by the press that printed this book. A likely place is the electronic control unit that monitors print density and automatically adjusts ink flow. The computer "reads" the print and decides whether to increase or decrease the ink flow to the paper. This decision must take into account various input parameters such as humidity, temperature, paper velocity, and specific gravity of the ink. The control algorithm written in machine code and stored in ROM shifts through all the input data and generates its conclusion in the form of a proportional output to an ink-flow valve.

In most cases, computerized functions do not stop with simple control. In any process where repeatability and quality control are important, significant process parameters are constantly monitored for deviation from preset limits and an alarm is set if the limits are exceeded. To aid in long-term analysis, the data acquisition function often includes recording raw-process data from the input sensors at specific intervals and gen-

THE REAL WORLD

I don't want to confuse you by discussing too many commercial applications of single-board controllers. I doubt there are many web presses hidden in closets to which you want to add computer control. There are, however, many equally challenging and less esoteric applications for computer controls around the home. For example, a few that come to mind include energy management, security, and environmental monitoring. I refer to such systems as real world systems, as opposed to the TTL digital world of computers.

Because real world is anything outside of the computer, it is generally an analog environment. The metamorphosis of ZAP into an intelligent controller is dependent primarily upon effective analog interfacing. For this reason, the rest of this chapter is dedicated to the design and construction of an economical analog I/O interface.

But first let's review the basics of D/A (digital-to-analog) conversion and then discuss a method to use a D/A to perform A/D (analog-to-digital) conversion. In data acquisition systems, there is often a need to acquire high resolution multiple channels, and AC as well as DC inputs. This being the case, I will also discuss a circuit which, in effect, allows ZAP to function as an 8-channel digital voltmeter. Finally, because the temporal relationship of so many events is significant, ZAP will be configured with a real-time clock that defines the time at which control operations occur.

DIGITAL-TO-ANALOG CONVERTERS

The D/A (digital-to-analog) converter can be thought of as a digitally controlled programmable potentiometer that produces an analog output. This output value (V_{OUT}) is the product of a digital signal (D) and an analog reference (V_{REF}) and is expressed by the following equation:

$$V_{OUT} = D V_{REF}$$

To a large extent, no D/A or A/D converter is very useful without specifying the type of code used to represent digital magnitude. Converters work with either unipolar or bipolar digital codes. Unipolar includes straight binary and binary coded decimal (BCD). Offset binary, one's or two's complement and Gray code, is usually reserved for bipolar operation. However, we will limit our discussion to straight and offset binary.

It is important to remember that the binary quantity presented by the computer is a representation of a fractional value to be multiplied by a reference voltage. In binary fractions, the MSB (most significant bit) has a value of $1/2$ or 2^{-1} , the next MSB is $1/4$ or 2^{-2} , and LSB (least significant bit) is $1/2^n$ or 2^{-n} (where n is the number of binary places to the right of the binary point). Adding up all the bits produces a value that approaches 1. (The more bits, the closer that value is to 1.) The algebraic difference between the binary value that approaches 1, and 1, is the quantization error of the digital system (to be discussed later).

Offset binary is similar to straight binary except that the binary number 0 is set to represent the maximum negative analog quantity; the MSB is a 0 for negative analog values, and a 1 for positive analog values.

The conversion of digital values to proportional analog values is accomplished by either of two basic conversion techniques: the weighted-resistor D/A converter and the R-2R D/A converter. The weighted-resistor D/A converter is by far the simplest and most straightforward. This parallel decoder requires only one resistor per bit and works as follows: switches are driven directly from the signals that represent the digital number D ; currents with magnitudes of $1/2$, $1/4$, $1/8$, . . . $1/2^n$ are generated by resistors with magnitudes of R , $2R$, $4R$, . . . $2^n R$, that are connected by means of switches between a reference voltage, $-V_{REF}$, and the summing point of an operational amplifier. The various currents are summed and converted to a voltage by an operational amplifier (see figure 8.1).

While this may appear to be a simple answer to an otherwise complex problem, this method has some potentially hazardous ramifications. The accuracy of this converter

is a function of the combined accuracies of the resistors, switches (all switches have some resistance), and the output amplifier. In conversion systems of greater than 10-bits resolution, the magnitudes of the resistors become exceptionally large and the resultant current flow is reduced to such a low value as to be lost in circuit thermal noise.

A reasonable alternative to the weighted-resistor D/A converter is the R-2R converter. This is often referred to as a resistor-ladder D/A converter and is the most widely used type even though it uses more components. This circuit (see figure 8.2) also contains a reference voltage, a set of binary switches, and an output amplifier. The basis of this converter is a ladder network constructed with two resistor values, R and 2R.

One resistor (2R) is in series with the bit switch, while the other (R) is in the summing line, so that the combination forms a "pi" network. This suggests that the impedances of the three branches of any node are equal, and that a current I, flowing into a node through one branch flows out as I/2 through the other two branches. In other words, a current produced by closing a bit switch is cut by half as it passes through each node on the way to the end of the ladder. Simply stated, the position of a switch, with respect to the point where the current is measured, determines the binary significance of the particular switch closure.

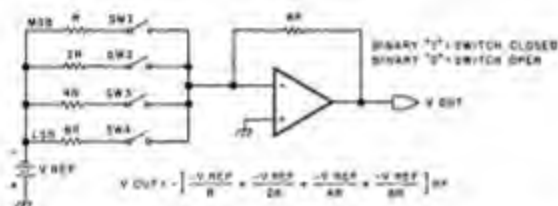


Figure 8.1 A 4-bit weighted-resistor digital-to-analog converter. A 4-bit word is used to control four single-pole single-throw switches. Each of those switches is in series with a resistor. The resistor values are related as powers of 2, as shown. The other sides of the switches are connected together at the summing point of an operational amplifier. Currents with magnitudes inversely proportional to the resistors are generated when the switches are closed. They are summed by the op amp and converted to a corresponding voltage.

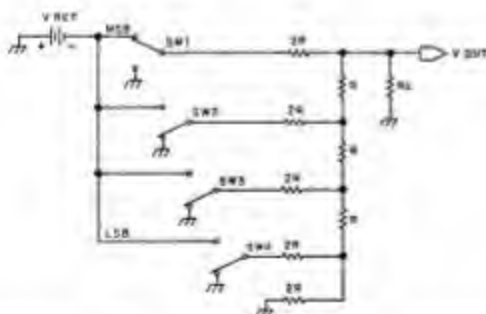


Figure 8.2 A 4-bit R-2R resistor ladder digital-to-analog converter. This type of D/A converter makes use of a resistor-ladder network constructed with resistors of value R and 2R. The topology of this network is such that the current flowing into any branch of a 3-branch node will divide itself equally through the two remaining branches. Because of this, the current will divide itself in half as it passes through each node on its way to the end of the ladder. The four switches are again related as powers of 2. The position of each switch with respect to its distance from the end of the ladder determines its binary significance.

This type of converter is easy to manufacture because only two resistor values are needed; in fact, one value, R , will suffice if three components are used for each bit. Keeping matched resistor values with the same temperature coefficients contributes to a very stable design. Certain trade-offs are required between ladder resistance values and current flow to balance accuracy and noise.

One form of the R - $2R$ ladder circuit is the multiplying D/A converter and is available with either a fixed or an externally variable reference. Multiplying D/A converters that utilize external variable analog references produce outputs that are directly proportional to the product of the digital input multiplied by this variable reference. These devices have either current or voltage output. The current output devices are much faster because they do not have output amplifiers that limit the bandwidth; therefore, they tend to cost less than voltage types.

An economical 8-bit multiplying D/A is the Motorola MC1408-8 (see figure 8.3). As previously mentioned, this monolithic converter contains an R - $2R$ ladder network and current switching logic. Each binary bit controls a switch that regulates the current flowing through the ladder. If an 8-bit digital input of 11000000 (192 decimal) is applied to the control lines of the illustrated converter, the output current would be equal to $(192/256)(2 \text{ mA})$ or 1.50 mA. Note that when binary 11111111 (255 decimal) is applied, there is always a remainder current that is equal to the LSB. This current is shunted to ground, and the maximum output current is $255/256$ of the reference amplifier current, or 1.992 mA for a 2.0 mA reference current. The relative accuracy for the MC1408-8 version is $\pm 1/2$ the LSB, or 0.19% of full scale (see figure 8.4). This is more than adequate for most home computer analog control applications.

The final circuit (figure 8.5) is an 8-bit MC1408-8 multiplying D/A converter. As previously outlined, "multiplying" means that it uses an external variable reference voltage. In this case, a 6.8 V zener-diode regulated voltage is passed through a resistor that sets the current flowing into pin 14 to approximately 2 mA.

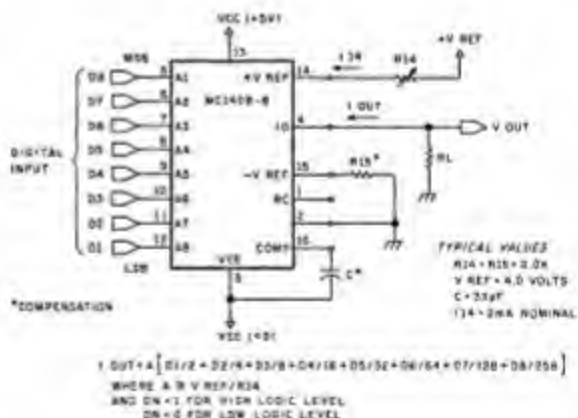


Figure 8.3 A typical 8-bit current-output monolithic multiplying D/A converter. This Motorola integrated circuit contains an R - $2R$ network like the one in figure 8.2, plus additional current-switching logic.

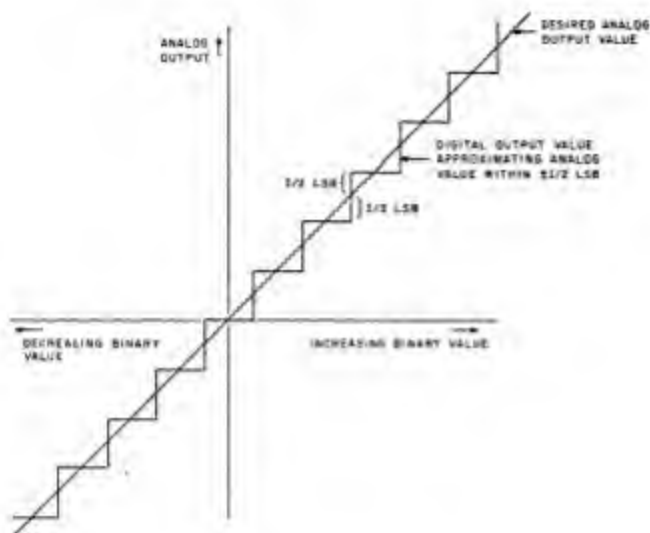


Figure 8.4 Output characteristics of a digital-to-analog converter showing least significant quantization.

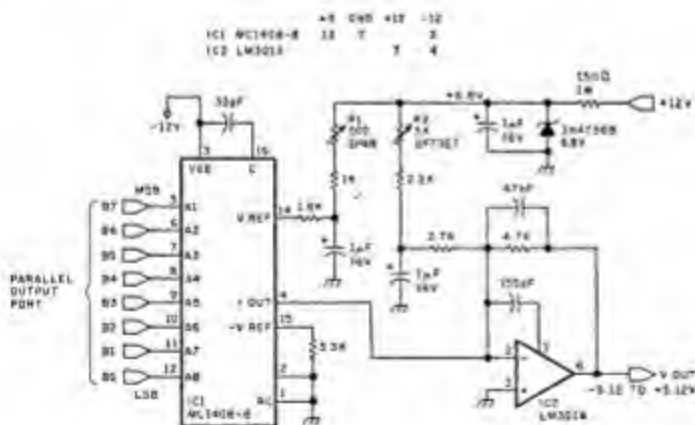


Figure 8.5 A final 8-bit MC1408-8 multiplying digital-to-analog converter with span and offset adjustments.

An additional resistor, R1 (also in this current leg), allows the current to be varied by a small percentage and provides the ability to adjust the full-scale range of the D/A converter. The output is a current that is equivalent to the product of this reference current and the binary data on the control lines. The current is converted to a voltage through IC 9 and can be zero offset through the use of the offset adjustment pot, R2.

Using this circuit with the ZAP computer is simply a matter of connecting the input lines of IC 1 to a convenient parallel output port on ZAP. Any 8-bit value sent to that port will be converted to a voltage proportioned to that output.

The digital code presented to the D/A converter must be in offset binary. A binary value of 00 hexadecimal produces an output of -5 V while FF hexadecimal is equivalent to +5 V. In offset binary, if the MSB is a 0, the output is negative, and if the MSB is a 1, the output is positive. Because the converter has a range of 10 V, and is an 8-bit device, the resolution of the converter is $1/256$ of 10 V, or approximately 40 mV. This means that the smallest output increments will be in 40 mV steps. To change this to finer increments requires a shorter range, such as +2.56 V to -2.56 V. By adjusting the span and zero pots, any reasonable range may be chosen, but the resolution will always be equal to the LSB or $1/256$ of the range, and accuracy is estimated to be $\pm 1/2$ the LSB.

Calibration is fairly straightforward. Apply the power, and with a short program that outputs a value from the accumulator, send a binary 10000000 to the port address corresponding to the D/A interface board. Using a meter to monitor the output of the LM301A, adjust the zero pot R2 until the output is 0 V. With the same program, load in binary 11111111 to the port address and adjust the span pot R1 for a meter reading of +5.12 V. A binary setting of 00000000 should produce -5.12 V. If you are unsuccessful at this point, turn the power off and remove the MC1408-8 and the LM301A; then reapply power and verify that the binary output is correct on the parallel output port. Nine times out of ten, problems like this can be attributed to choosing an incorrect output code.

If the test is successful, you are now ready to generate analog outputs under program control. A simple test is to designate a section of memory and sequentially output the values to the D/A. If the table is 256 bytes long with the values ranging from 0 to FF hexadecimal in 01 increments, the result will be a sawtooth-waveform output. If the samples are sent to the output rapidly enough, and it is connected to a speaker, the waveform will be audible. The exact frequency will be a function of the update timing loop.

The following is a short program that exercises the D/A in such a manner:

START	EQU	0400	Memory table start HL address
END	EQU	05	Memory table end H address
OPORT	EQU	07	D/A output port number
SAMP	EQU	A0	Sample rate time constant
AGAIN	LD	HL, START	Load table start address
	LD	A, (HL)	Table value to accumulator
	OUT	OPORT, A	Output byte to D/A
	CALL	DELY	Sample time delay
	INC	HL	
	LD	A, H	
	CP	END	Test to see if at end of table
	JP	NZ, AGAIN	If not, output the next sample
	HALT		
DELY	LD	B, SAMP	Sample rate timing loop
DCR	DEC	B	
	JP	NZ, DCR	
	RET		

The table can be set to any length. Values in the table can be calculated to produce any shape waveform.

ANALOG-TO-DIGITAL CONVERTERS

It's always a good idea to discuss D/A converters first. They are rather straightforward and there are not an overwhelming number of conversion methods. By introducing them first, you will become aware of the process of binary conversion and appreciate the concepts of resolution and accuracy. Practically speaking, however, if you were going to set up the ZAP computer to serve in a data acquisition mode—say, reading and recording temperatures—you would need an A/D (analog-to-digital) converter before a D/A (digital-to-analog).

An A/D does what its name implies. It converts analog voltages into a digital representation compatible with the computer input. As in the case of an 8-bit D/A, an A/D is subject to the same conversion rules. If you are trying to read a 10 V signal with an 8-bit converter, the resolution will be $1/256$ of 10 V (or 40 mV) and the accuracy will be $\pm 1/2$ the LSB.

For greater resolution more bits are necessary. The number of bits does not set the range of a converter; it only determines how finely the value is represented. An 8-bit converter (either A/D or D/A) can be set up just as easily to cover a range of 0 to 1 V or 0 to 1000 V. Often the same circuitry is used, but a final amplification stage or resistor-divider network is changed. Understand, of course, that with a range of 1000 V and an 8-bit converter, the resolution is 4 V. Such a unit would be useless on 0 to 10 V signals. The problem can be reconciled in a number of ways, but the easiest solution is to use a converter with more bits. A 16-bit converter that has 65,536 (2^{16}) steps instead of 256 (2^8) would cover the same 1000 V range in 15 mV increments.

For the ZAP computer, the question becomes more one of reasonable price performance than nth degree accuracy.

Analog-to-digital conversion is considerably more expensive than D/A—the price is directly related to resolution and accuracy. There are many ways that A/D conversion can be accomplished. The range varies from very slow, inexpensive techniques to ultrafast, expensive ones. An A/D converter can cost as little as \$5 or as much as \$10,000. An A/D converter that scans thermistor probes and provides data to control the temperature in a large supermarket may cost \$4.75, but it cannot encode video information from an optical scanner.

The objective of this book, of course, is to help you to build your own computer; little is served by presenting designs that are beyond a reasonable budget and average construction abilities. For those reasons, I have sifted through a multitude of techniques to select four designs that can easily be built and attached through the ZAP computer's parallel interface. One of them should meet your basic data acquisition requirements.

1. Basic analog to pulse width converter
2. Low cost and low speed 8-bit binary-ramp counter converter
3. High speed 8-bit successive approximation converter
4. Eight-channel $3\frac{1}{2}$ -digit 0–200 V AC/DC interface

PULSE WIDTH AND BINARY COUNTER CONVERTERS

Analog to Pulse Width Converter

This converter is one of the most popular open-loop encoders because of its simplicity. A basic block diagram is shown in figure 8.6. This device uses a fixed oscillator in combination with a circuit that generates a pulse width that is a linear function of the analog input voltage.

To obtain this variable linear pulse width, designers frequently use a ramp generator and a Schmitt-trigger circuit. A gating pulse is started at the beginning of the ramp and a counting circuit starts incrementing at a fixed frequency. When the linear ramp reaches the same value as the input voltage, the counting is terminated. The value left in the register at that point is representative of the analog input.

Figure 8.7 is a schematic of a unipolar analog to pulse width converter that operates on this principle. IC 1 is configured as a gate controlled linear ramp generator and IC 2 is the input comparator. The process starts when the 7.5 KHz clock signal fires IC 3 (a 74121 one-shot), and starts its 35 ms period, which is the gate time. At the beginning of this gate period, a pulse that clears the two 7493s and the ramp generator is generated.

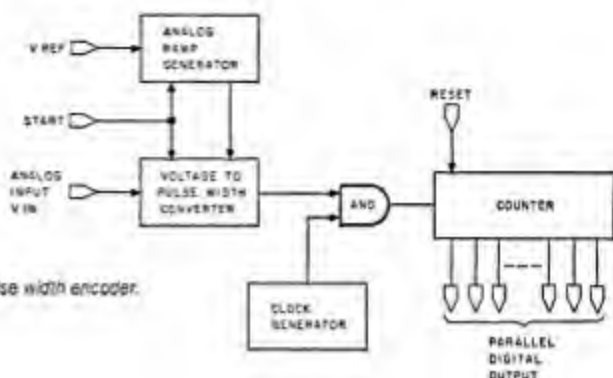
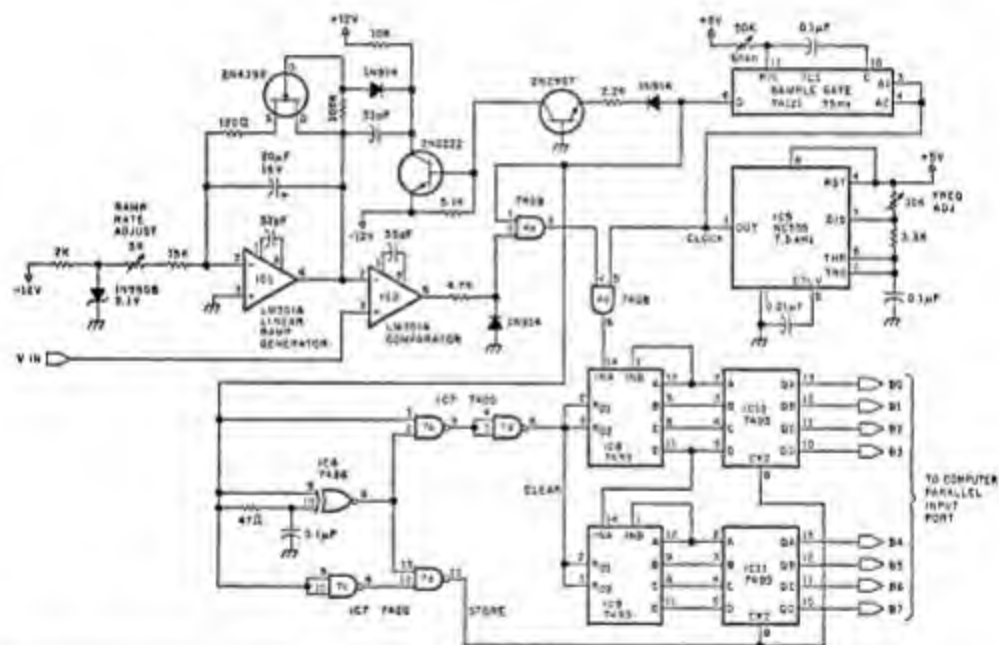


Figure 8.6 A block diagram of an analog to pulse width encoder.



IC#	TYPE	+5V	GND	+12V	-12V
1	LM201A			7	4
2	LM201A	7			4
3	74121	14	7		
4	7408	14	7		
5	NE555	8	1		
6	7486	14	7		
7	7800	14	7		
8	7493	5	10		
9	7493	5	10		
10	7493	14	7		
11	7493	14	7		

NOTES: 1. SET WAMP TO GO FROM 0 V TO FULL SCALE DURING SAMPLE GATE TIME

2 SET FREQ TO PRODUCE WHATEVER COUNT IS DESIRED TO REPRESENT INPUT VOLTAGE
ie 256 COUNTS DURING SAMPLE PERIOD FOR 2.56 VOLTS.

Figure 8.7 A schematic diagram of a unipolar analog to pulse width converter.

This, in turn, enables the clock signal to the counter. The slow rate of the ramp generator is set to be approximately 10 V per 35 ms. IC 2 continuously compares the input and ramp voltages. When they are equal, the clock signal to the counter is stopped and the ramp generator is reset. At the conclusion of the 35 ms gate time, whatever value is in the counter is transferred to an 8-bit storage register. The value stored in this register is an 8-bit number proportional to the input voltage. The entire process starts again on the next clock pulse.

By properly selecting the gate times and the clock rate, you can change the span and resolution of the circuit. With a gate time of 35 ms and a clock rate of approximately 7500 Hz, 256 clock pulses should be counted during the gate time. The ramp timing adjustment pot should be set so that the counter reaches maximum count when 2.56 V is applied to the input of IC 2. A 10:1 divider attached to this input will allow the same 8-bit count to represent 25.6 V.

This circuit is simple, but its accuracy depends on the stability of the individual sections of the circuit. To use it, connect the register output to a parallel input port. Simply read the port when you want the latest value. The circuit automatically updates 28 times a second, hence no reading is older than 35 ms.

Binary-Ramp Counter Converter

The above A/D technique is most often used in slow sampling rate, high-accuracy measurements. Achieving these results, however, hinges on the use of precision components and proper construction. The next most productive approach to consider is the binary-ramp counter method. In my opinion, this is the best type if you plan to construct an A/D for ZAP. It uses fewer components and, in practice, is much faster and easier to build than linear-ramp circuits.

Figure 8.8 illustrates the basic block diagram for the binary-ramp counter converter. The linear-ramp generator of the previous technique has been replaced by a D/A converter. In this case, the D/A is used to reconvert the digital output of the binary counter back to analog for comparison against the analog input. If they are equal, then whatever code is presently set on the D/A input is also our A/D output.

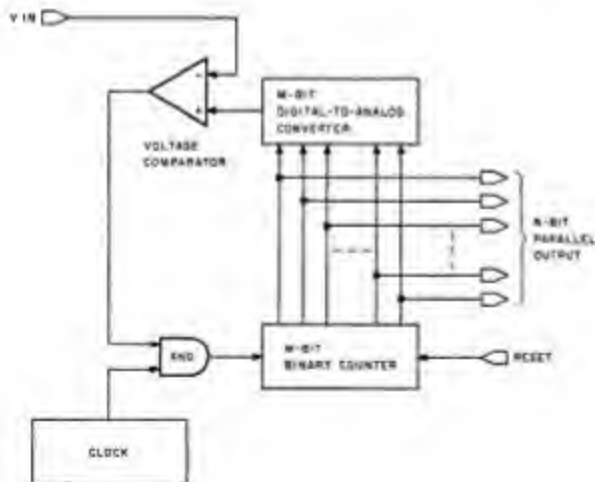
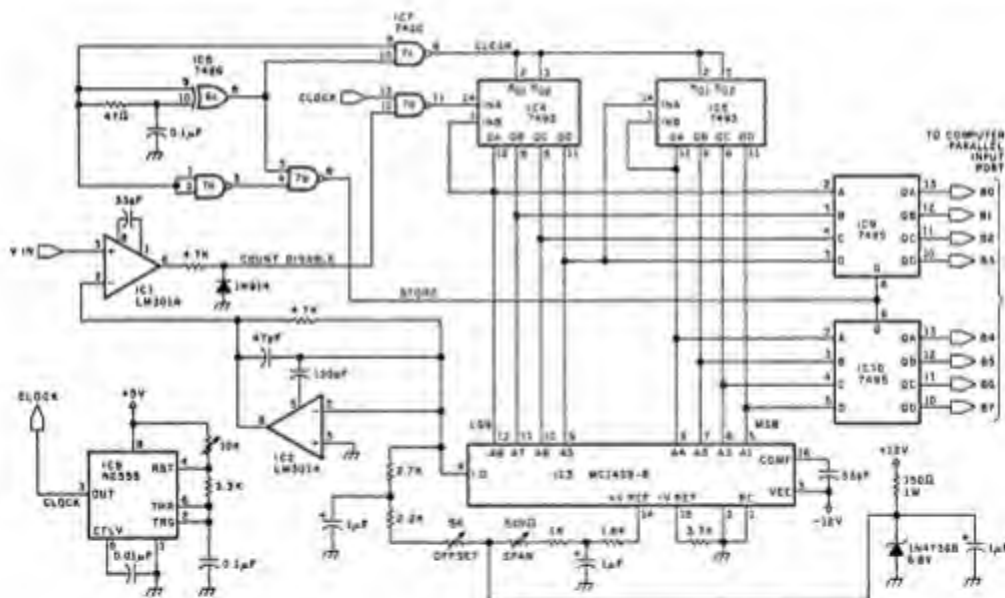


Figure 8.8 A block diagram of a basic binary-ramp counter A/D converter.

The simplest way to operate the system is to start the counter initially at 0 and to allow it to count until the D/A equals or exceeds the analog input. The only critical consideration in designing this circuit is that the clock rate cannot be faster than the response of the comparator and D/A. If it takes 100 μ s for these components to do their job, then the maximum clock rate should be 10 KHz. For an 8-bit converter (counting from 0 to 256 each sample period), the maximum sample rate is 10,000/256 or some 39 samples a second. In practice, however, 5 μ s is a more reasonable settling time, resulting in about 750 samples per second. For still higher speeds, we use a different kind of A/D (more on this later).

Figure 8.9 shows a schematic of a binary-ramp counter converter that uses a MC1408-8 multiplying D/A converter chip. The counter output is connected to the MC1408-8 to provide a direct analog feedback comparison of the value set on the counter. Initially, ICs 4 and 5 are cleared, and the D/A output should equal the minimum input voltage. For a 0 to 5.12 V converter, this would be 0 V. For a -2.56 to +2.56 V unit, it would be -2.56 V. If the output of IC 1 is less than V_{in} , the clock pulses are allowed to reach the counter. As each pulse increments the counter, the output of the D/A keeps rising until eventually it equals or just exceeds V_{in} on the comparator. When this happens, additional clock pulses are inhibited. At the end of the sample period, the count value of ICs 4 and 5 is stored in a separate register. For ZAP to read this data, it just requires connecting this register to an input port and reading it directly.



IC #	TYPE	+5V	50K	+12V	-12V
1	LM301A	7		4	
2	LM301A		7	4	
3	MC1408-8	13	7		3
4	7493	8	10		
5	7493	8	10		
6	7495	14	7		
7	7495	14	7		
8	7495	8	1		
9	7495	14	7		
10	7495	14	7		

Figure 8.9 A schematic diagram of an 8-bit binary-ramp counter A/D converter.

Using the Computer to Replace the Counter

Figure 8.9 is a stand-alone circuit. It does not require the computer for operation. The A/D updates itself at a preselected sample rate and loads this value into an 8-bit latch. As far as the computer is concerned, there is a steady state reading from the converter. Every function required to perform the A/D conversion is constructed from hardware components.

There are certain advantages to this approach. The A/D can be independently assembled and tested without a computer. For example, a voltage can be applied to the input and the 8-bit value can be displayed on 8 LEDs. The ability to test each subsystem independently is the way I've tried to present all the hardware in this book. If, on the other hand, you feel you've mastered the art of programming and would rather not build elaborate interfaces, much of the hardware of figure 8.9 can be replaced with software subroutines.

Consider for a moment the major elements of this design. This 8-bit A/D has four sections: D/A, analog comparator, 8-bit counter, and timing logic. The resistor ladder and analog comparator are necessary components, but the last two sections are prime candidates for synthesis through the computer. The combined function of these devices is to increment an 8-bit count and check the output of the comparator.

The ZAP computer has parallel input and output ports. By incrementing a central processor register and outputting the value after each increment, the 8 lines from the port will have all the appearances of a standard 8-bit counter made with 7493s and so on. By using one bit of an input port to read the status of the comparator, we can also replace the rest of the timing logic.

The resulting interface has fewer components and is shown in figure 8.10. The D/A remains essentially the same except that rather than being driven from two 4-bit counters, it is connected to an 8-bit parallel output port. The analog output of the D/A will be whatever value is sent to the output port. Instead of hardwired logic to detect when the D/A and input voltage are equal, we attach the comparator output to bit 0 of an available input port.

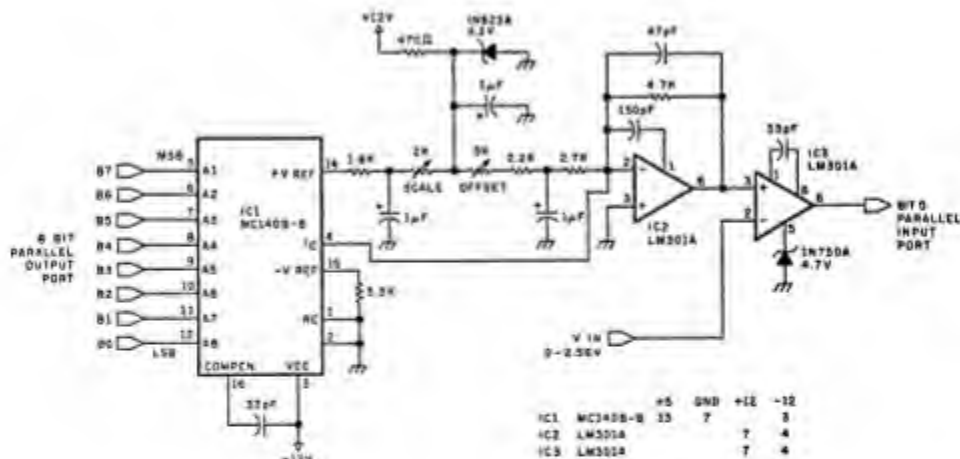


Figure 8.10 A software-driven 8-bit analog-to-digital converter.

The conversion process is not unlike the hardware version. First, we clear a register (B, for example) and then output the register value to the port attached to the D/A. This will set the D/A to its minimum output. Next, we read the input port that has the comparator attached to it and check bit 0 (a logic 1 indicates that the input and D/A voltages are equal). If the comparator is low (the voltages are not equal), the register is then incremented and the process is repeated. Eventually, the register will be incremented to the point where the D/A output and the unknown input voltage are equal. The comparator will then switch. At this point the program is halted and the value of the B register is the digital equivalent of the input voltage. The program to accomplish this follows:

	MVI	B	Clear B register
	OUT	0, B	Output B register
AGAIN	INC	B	Increment B register
	OUT	0, B	Output B register
	IN	04	Read comparator port
	ANA	01	Isolate bit 0
	JNZ	AGAIN	Continue if voltages not equal
	HLT		A/D value is in B register

The above program should be repeated each time a new reading is needed and the sample rate can be adjusted within broad limits. Remember, however, that we still have to wait for the D/A circuitry to settle and it should not be incremented any faster than 5 μ s. Using the 2.5 MHz Z80 should not present a problem. Using a 4 MHz crystal the central processor might necessitate a few NOPs in the loop.

There are many variations on this circuit. As described, it takes up to 255 iterations of the program to find an answer. On a computer with a 2 μ s average instruction time, the program could take 3 μ s to finish, limiting us to about 300 samples a second. Add the other tasks that the computer must perform and you might be limited to 100 samples a second. Executing counting routines takes time; it will not, however, be a problem if you are merely monitoring a temperature probe that has a 30-second time constant.

If you should want to track and record fast changing signals, such as an acoustic waveform, then a much faster conversion algorithm is required. One method that speeds up the process is called successive approximation (more later).

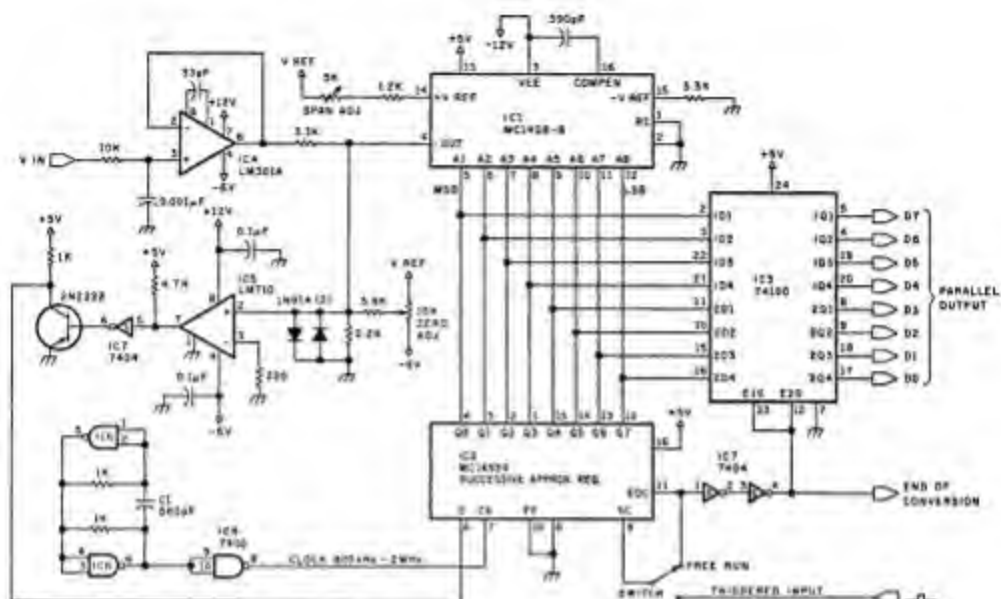
The capabilities of this circuit can be expanded in other ways. An additional CMOS multiplexer can be connected to 3 bits of another output port to turn this simple circuit into an 8-channel A/D. Also, because this circuit includes a D/A, its output is available as well.

Successive Approximation Converters

More than likely one of the three converters presented thus far will suffice for non-critical data acquisition. Slowly changing signals can be handled accurately and efficiently. However, there are occasions when the signal in question is not slow or it carries a particular transient that must be captured. For example, detecting a 100 μ s event requires a converter with a capability of 20,000 samples per second. In such cases we need a much faster conversion method.

Figure 8.11 is the schematic of a general purpose high-speed, 8-bit converter. It is capable of sample rates in excess of 200,000 samples per second. To attain these speeds, a technique called successive approximation is used. Like the binary-ramp counter converter, this A/D also incorporates a D/A in a feedback loop but replaces the counters with a special SAR (Successive Approximation Register). The circular logic of successive approximation is best explained in the block diagram of figure 8.12.

Initially the output of the SAR and mutually connected D/A are at a zero level. After a start conversion pulse, the SAR enables the bits of the D/A one at a time starting with the MSB. As each bit is enabled, the comparator gives an output signifying that the input signal is greater or less in amplitude than the output of the D/A. If the D/A output is greater than the input signal, a "0" is set on that particular bit. If it is less than the input signal, it will set that bit to "1". The register successively moves to the next least



IC#	TYPE	+5V	GND	+12V	-12V	-6V
1	MC1458	18	5			
2	MC1458	18	8			
3	74100	24	7			
4	LM301A		7			
5	LM710		8			
6	7400	18	7			
7	7404	18	7			
8	MC1458	18	1			
9	LM301A		7			

- NOTES:
1. ALL RESISTORS ARE 1/4W 5% UNLESS OTHERWISE INDICATED.
 2. ALL CAPACITORS ARE 100V CERAMIC UNLESS OTHERWISE INDICATED.
 3. WITH COMPONENTS SHOWN, CLOCK FREQUENCY IS 800 KHz. THIS IS 100,000 CONVERSIONS PER SECOND IN FREE RUN MODE.
 4. THE FOLLOWING CIRCUIT CAN BE ADDED TO EACH OUTPUT PIN OF IC8 IF A VISUAL INDICATOR IS DESIRED.

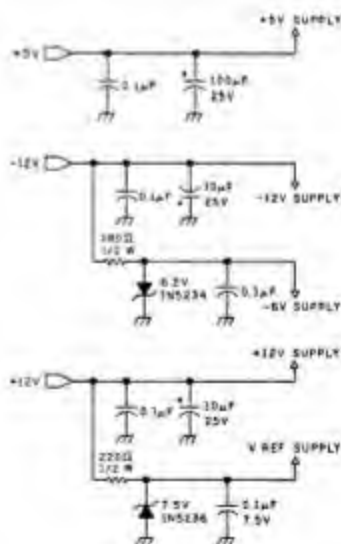
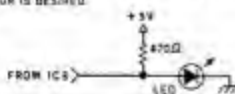


Figure 8.11 A schematic diagram of an 8-bit successive approximation A/D converter.

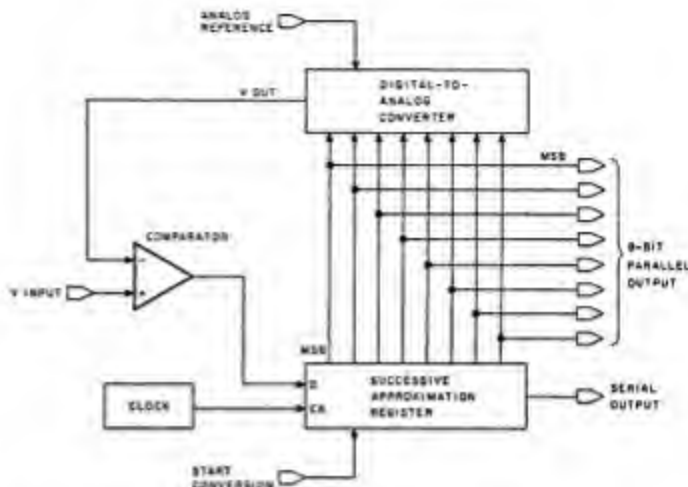


Figure 8.12 A block diagram of a typical 8-bit successive approximation A/D conversion system.

significant bit (retaining the setting on the previously tested bits) and performs the same test. After all the bits of the D/A have been tried, the conversion cycle is complete. As opposed to the 256 clock pulses of the binary counter method, the entire conversion period takes only 8 clock cycles. Another conversion would commence on the next clock cycle when it's in the free-run mode. To retain the 8-bit value between conversions, an 8-bit storage register IC 3 has been added. To use this A/D, simply connect the output of this latch to an 8-bit input port.

The components of the D/A circuit are changed slightly from previous implementations to increase the speed, and a faster comparator is used. With a clock rate of 800,000 Hz, the circuit will do 100,000 conversions a second. Because they are automatically loaded into the 8-bit-holding register IC 3, the update is transparent to the computer and can be read at any speed. The sample rate is a function of the clock rate. If it is unnecessary to have such a high sample rate, it may be reduced by increasing the value of C1. High speed A/D converters are susceptible to layout and component selection. While 200,000 samples per second is attainable, 20,000 samples per second might be more practical.

A Unique Application for a Fast A/D

When we first considered adding an A/D to ZAP, our thoughts centered on monitoring some process or turning ZAP into an intelligent controller. In most cases, this requires one of the simpler A/D converters I've outlined. However, with the addition of a high speed A/D peripheral, a few more experiments come to mind.

Most often when we think of high speed analog, we want to capture video or other high bandwidth phenomena that have a voltage level within the range of the A/D. Of course, the audio frequencies, while much lower than video, may also require a high performance A/D for proper representation.

The bandwidth of the human voice is about 4000 Hz. These analog signals, when spoken into a microphone and fed to an A/D, can be digitized just like any other waveform. And, if our voice samples are taken quickly enough and stored, the accumulated data can be used to reconstruct the same voice. This reconstructed voice is called digitized speech.

In essence, digitized speech is simply the result of a standard data acquisition technique. When speaking into a microphone and amplifier, your voice results in a fluctu-

ating waveform, whose frequency rate varies. If this signal is applied to the input of a high speed A/D, and the conversions stored in memory, the computer couldn't care whether the source was speech or a nuclear reaction. The analog fluctuations would be digitized at discrete sampling intervals and stored. If the stored samples are output to a D/A at the same rate they were taken, speech will be reproduced. The fidelity of this reconversion is a function of the sampling rate.

Most of the intelligence or information content of human speech occurs in the frequency region below 1500 Hz. Obviously, sampling this waveform at 25 samples per second would be useless. It must be sampled very rapidly to retain anything of significance.

There is a specific law known as the "Nyquist criterion" that is used to determine the optimal sampling rate. In theory, this law states that at the very minimum, the sample rate must be twice the frequency of the input waveform. Thus, if the human voice extends to 4 Hz, then the minimum rate should be 8000 samples per second. This also presumes an ideal filter on the output, the existence of which is about as ephemeral as perpetual motion. In actuality, the sampling rate should be 3 or 4 times the highest input frequency. To digitize voice accurately requires a sampling rate of 12 Hz to 16 Hz. If, on the other hand, we shoot for just the lower frequencies, we can get by with 3 Hz or 4 Hz.

The possibility of using this speech technique has to be considered in light of the availability of large amounts of memory. At a 4 Hz sample rate, one second of speech takes 4000 bytes of memory. If you have added more than the 2 K of memory in the original configuration of ZAP, then perhaps you'll want to experiment with digitized speech. Even with just 2 K you should hear something.

A fairly simple program is needed to coordinate the digitization process and store the data:

START	EQU	400	Memory table start HL address
END	EQU	C00	Memory table end H address
TRIG	EQU	A8	Input start conversion level
IPOINT	EQU	04	A/D input port
SAMP	EQU	38	Sample-rate time constant
AGAIN	INP	IN	Read A/D input value
	CP	TRIG	Compare input to trigger level
	JP	NZ, INP	Loop again if below trigger level
	LD	HL, START	Load table start address
	IN	IPOINT	Take a sample
	LD	(HL), A	Store sample in memory
	CALL	DELY	Delay between samples
	INC	HL	
	LD	A, H	
	CP	END	Test to see if at end of table
	JP	NZ, AGAIN	If not, take another sample
	HALT		
DELY	LD	B, SAMP	Start delay timer
DCR	DEC	B	
	JP	NZ, DCR	
	RET		

When the program is executed, it will scan the A/D input port and compare the reading to A8 hexadecimal (about 65% of full scale). When speech is present, the audio level will presumably exceed this trigger level. When this happens, the program sets the address of the storage table and starts dumping data samples into it at a rate of about 4000 per second. The rate is determined by the value of "SAMP." The higher the number, the lower the sampling frequency. When the table is filled, the program stops and the memory will contain a digitized representation of whatever was spoken during the sample time. For 2 K of memory, only 1/2 second of speech will be captured.

To hear this stored data, use the program outlined in the section on D/A converters.

Set the limits to be the area of the memory table, then choose a time constant that results in putting out the samples at the same rate that they were taken. (It is also possible to create a digital reverberation system using this hardware, but for decent fidelity 12- or 14-bit converters are required.)

Because digitized speech is a specialized application, the D/A circuit is modified slightly to include a low-pass filter. This will improve the sound quality. The modified circuit is shown in figure 8.13.

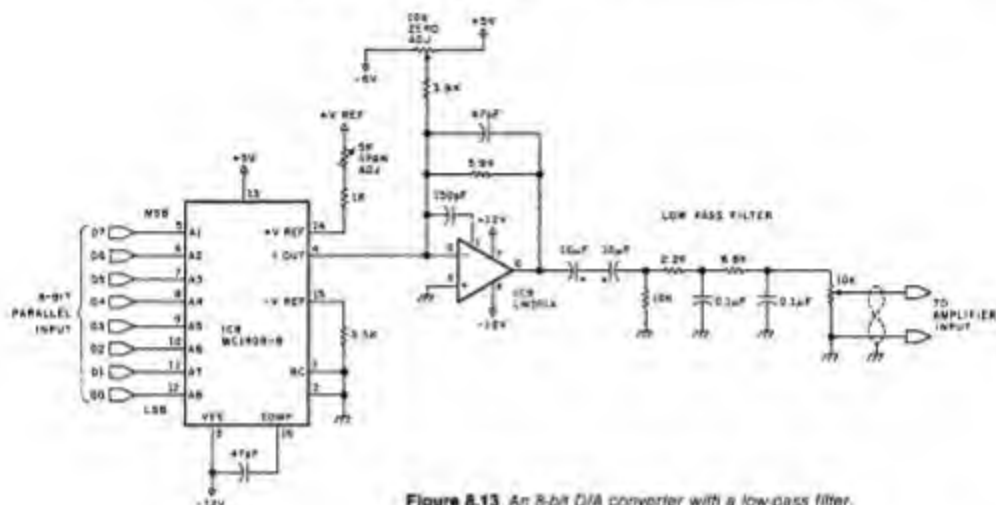


Figure 8.13 An 8-bit D/A converter with a low-pass filter.

Using ZAP for High Resolution Data Acquisition

Up to this point our discussion has concerned experimenting with ZAP. Some aspects of these designs are useful in noneducational applications, but for the most part they are intended more as teaching aids than as replacements for expensive monitoring equipment. However, it is possible to add more specialized interfacing to ZAP which allows it to be used in such a manner.

The 8-bit A/D converters presented thus far have limited resolution and are single-channel devices. They are adequate for measuring temperature in a solar heating system, but it is doubtful that they have the resolution to monitor the temperature gradient along a length of heating duct. The sensors used to measure such parameters would need to have a higher resolution than ambient air temperature sensors. For a range of -20 to 108°C , an 8-bit A/D could provide 0.5° resolution. In a solar heating application, considering the variations in air movement, cloud cover, and general weather patterns, this is as much resolution as you would need. Within the system, however, there are areas that will require closer measurement.

A solar system is a typical example. After installation the next step is usually to investigate how to increase its efficiency. Nine times out of ten this requires cutting heat losses in the pipes and ducts. One way to determine such loss is to place temperature sensors along the heat distribution path and look for cold spots. The measured differences between sensors may be very small, a few tenths of a degree or so, but the overall losses could be significant. Measuring temperatures to tenths or hundredths of a degree and maintaining the same dynamic range requires more than 8-bit resolution. Something between 10 and 12 bits is needed.

The situation is further complicated by the large number of points that may need monitoring within a system. It's rare to find only one temperature indicator in the system. At the very least there would be six: inside air, outside air, storage tank top, storage tank bottom, collector, and distribution air temperature.

Very few commercial data acquisition systems use a single channel. Usually they come with either eight or 16 multiplexed channels. The input of one A/D converter is switched (usually on a demand basis) between the channels and the results are compiled and averaged by the computer. This information can be logged on recording tape, transmitted serially to another system, or used to run a real-time display. What one does with the data is a function of the application program.

There are various ways to configure ZAP for high-resolution data acquisition. One is to simply to replace the 8-bit A/D with a 12-bit binary converter. When the conversion is finished, 12 bits of parallel data are available. Depending upon the converter chosen, many outboard analog components might still be required, but the process is straightforward. Unfortunately, these converters are not what you would call inexpensive. Although they are becoming cheaper every day, at this writing they are still considerably more expensive than 8-bit converters of similar speed.

Most 12-bit binary converters are expensive because they are designed to give the appearance of parallel converters. Toggle the convert enable line and zip, there's 12 bits of answer. When the computer wants this data, it scans, manipulates, and stores it in a table for use by other programs. Making the hardware section of an A/D interface less expensive involves doing less in parallel. Taking the alternative serial approach generally requires more time and additional data manipulation. We can opt for the lowest expense and let our computer do most of the work. We have already demonstrated how to eliminate counters and timing logic by doing these functions in software.

An 8-Channel 3½-Digit AC/DC Interface for ZAP

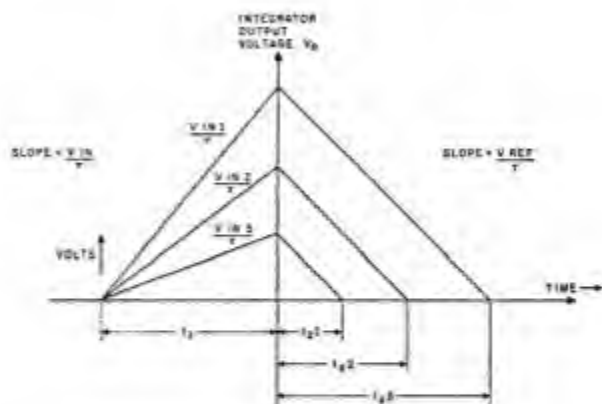
The solution to the high resolution versus expense question comes in the form of a 3½-digit multiplexed A/D converter chip. The MC14433 CMOS integrated circuit is intended primarily for use in digital voltmeters (DVMs) but enjoys a variety of other applications because of its versatility. It is a single-channel 11-bit converter, but it is called 3½ digits. The output is BCD (binary-coded decimal) and it specifically covers a range of -1999 to +1999 counts. Basic chip specifications are as follows:

MC14433 3½-Digit A/D Converter

Accuracy: $\pm 0.05\%$ of reading ± 1 count
Two voltage ranges: 1.999 V and 199.9 mV
25 conversions per second
1000 M Ω input impedance
Auto zero
Auto polarity
Over, under, and auto ranging signals available

The MC14433 is a modified dual-ramp integrating A/D converter and is outlined in figure 8.14. The conversion sequence is divided into two integration periods: unknown and reference. During the V_{in} (unknown input) integration sequence, the unknown voltage is applied to an integrator with a defined integration time constant for a predetermined time limit. The voltage output of the integrator then becomes a function of the unknown input input. The more positive the input, the higher the integrator output.

During the second cycle of the integration sequence, a reference signal of 2.000 V is connected to V_{in} . This causes the integrator to move toward zero while the digital circuitry of the chip keeps track of the time it takes to reach zero. The time difference between the two integration sequences is then a function of their voltage difference. If 2.000 V were the applied V_{in} then t_2 would equal t_1 . The unknown voltage is equivalent to the ratio of the periods times the voltage reference (V_{REF}). This is also known as a ratiometric converter. The full scale of the converter is determined by V_{REF} . Changing V_{REF} to 0.200 V will make the 1999 count output represent 199.9 mV instead of 1.999 V full scale.



r = INTEGRATION TIME CONSTANT
 t_1 = UNKNOWN VOLTAGE INTEGRATION PERIOD (CONSTANT)
 t_2 = REFERENCE VOLTAGE INTEGRATION PERIOD (VARIABLE)

$$V_O = \frac{V_{IN}}{r} t_1 + \frac{V_{REF}}{r} t_2$$

WHAT IS

$$\frac{V_{IN}}{V_{REF}} = \frac{t_2}{t_1}$$

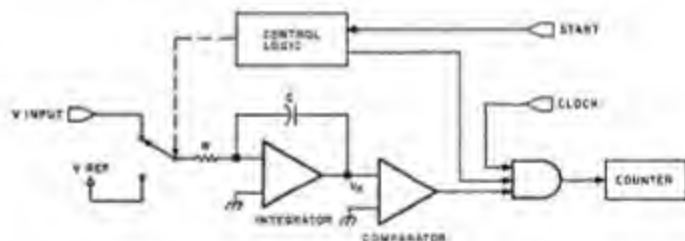


Figure 8.14 A simplified representation of a dual-ramp A/D converter.

The output of the DVM chip is a combination of serial and parallel data. There are 4 digit-select and 4 BCD data lines:

BCD Output Lines

Pin 23	Q3 (MSB)
Pin 22	Q2
Pin 21	Q1
Pin 20	Q0

Digit-Select Outputs

Pin 19	DS1 (MSD)
Pin 18	DS2
Pin 17	DS1
Pin 16	DS0

With respect to what the computer sees through 74LS04 output buffers, the digit select output is low when the respective digit is selected. The most significant digit ($\frac{1}{2}$ DS1) goes low immediately after an EOC (end-of-conversion) pulse and is followed by the remaining digits in a sequence from MSD to LSD. The multiplex clock rate is the system clock divided by 80; two clock periods are inserted between digit outputs.

During DS1, the polarity and certain status bits are available. The polarity is on Q2 and the $\frac{1}{2}$ digit value is at Q3. If Q2 is a "1", then the input voltage is negative, and if Q3 is a "0", then the $\frac{1}{2}$ digit is a 0.

Figure 8.15 details the schematic of the 8-channel interface board. As shown, it has the following capabilities:

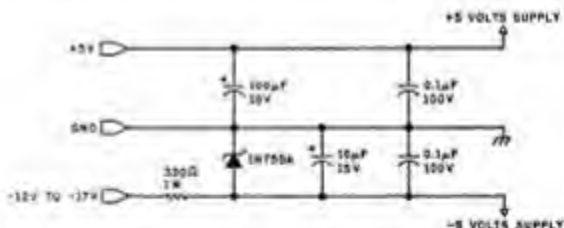
ZAP 3 $\frac{1}{2}$ -Digit DVM Interface

- 8 programmable-input channels
- AC or DC input capability
- Programmable gain of 1, 10, or 100
- Ranges of 0–200 mV, 0–2 V, 0–20 V, or 0–200 V
- Input overvoltage protection

IC 1 is the MC14433 DVM chip. It is set for approximately 25 conversions a second and all outputs are buffered. IC 2 is a precision voltage reference chip that supplies the V_{REF} signal. It is nominally 2.5 V and is trimmed to 2.000 V and 0.200 V with two potentiometers. While a zener diode might provide the same voltage, the temperature drift associated with such components makes them inadvisable in this application.

IC 5 is configured as a set/reset flip-flop. When the conversion is finished, an EOC signal sets IC 5, indicating to the computer that data is available. When the computer finishes reading the data, it resets this flip-flop and awaits the next conversion.

ICs 1, 2, 3, and 4 constitute a single-channel 3 $\frac{1}{2}$ -digit converter. It has a range of either 0.200 V or 2.000 V determined by V_{REF} . To achieve multichannel operation and AC capability, it is necessary to place an input multiplexer and AC to DC converter in front of IC 1.



1. ALL RESISTORS ARE 5% 1/4W UNLESS OTHERWISE NOTED
2. ALL CAPACITORS ARE 100 V CERAMIC UNLESS OTHERWISE NOTED

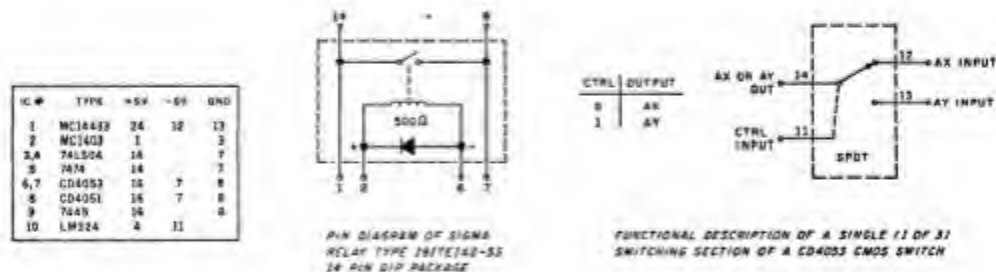


Figure 8.15 An 8-channel 3 $\frac{1}{2}$ -digit 0–200 V AC/DC DVM interface (continued on next page).

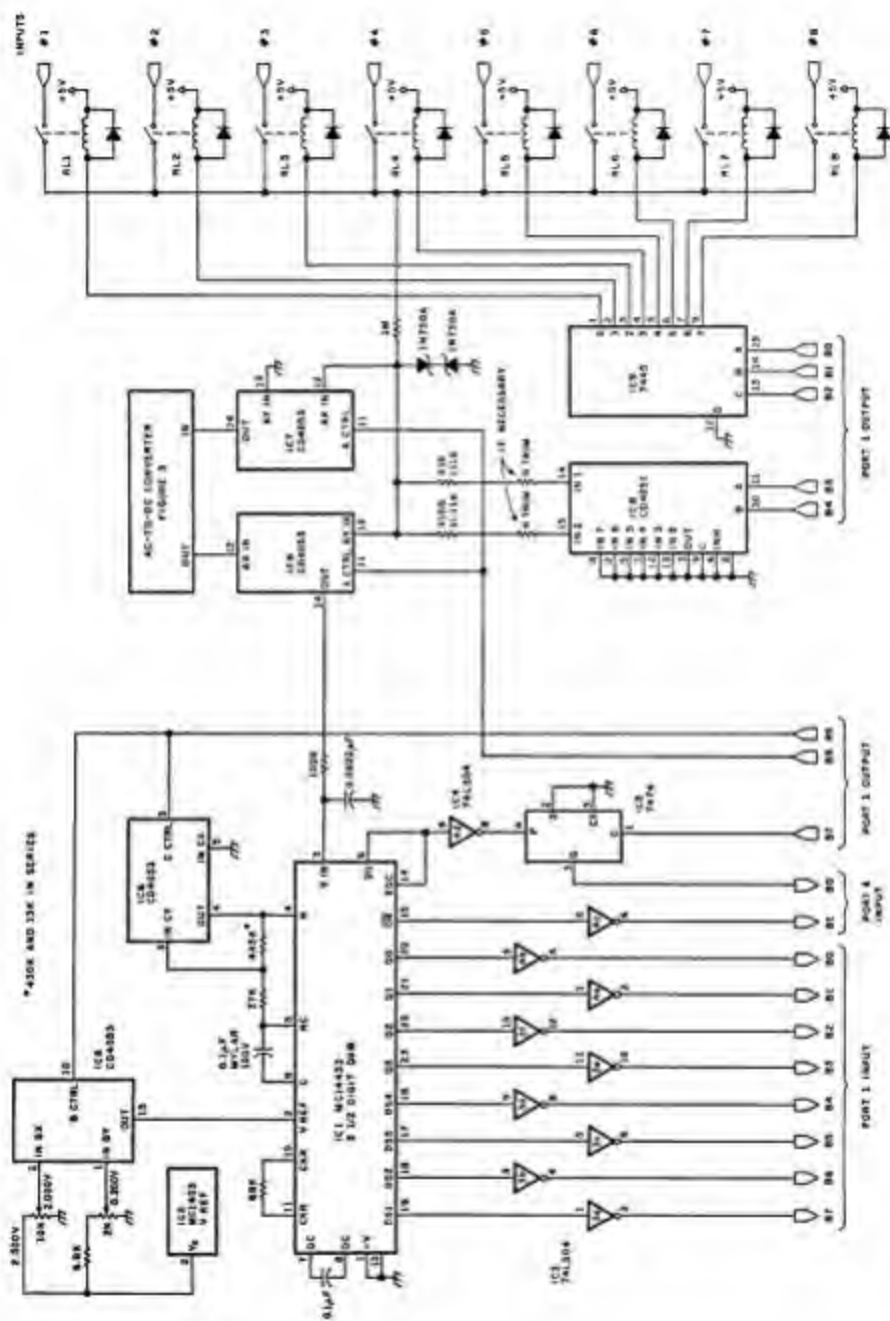


Figure 8.15 continued

Figure 8.16 shows the voltage reference and range selection setup of this interface. The MC14433 can cover either 0–199.9 mV or 0–1.999 V. The ranges depend upon the level of V_{REF} . When B5 of port 1 is low, switches 5 and 6 are in the positions shown. This would apply 2.000 V to V_{REF} input and set the integration time constant with an 82 k Ω resistor. With B5=0, V_{REF} is 0.200 V, and the integration resistor is 10 k Ω .

Figure 8.17 illustrates the input subsystem in simplified terms. SW1 and SW2 represent the gain selection section. As shown, the gain is 1 and no divider network is enabled. When an input relay is closed (controlled through IC 9), the input voltage of that channel is sent directly to the input of IC 1 through a 1 M Ω resistor. If the interface is set for DC and a gain of 1, a 1.400 V input signal at channel 3 would be read directly as 1.400 V by the DVM chip. If, however, 150 V were suddenly applied, it would be shunted through Z1 and Z2, which protect IC 1. The data read by the computer will indicate an out of range condition because the input would be shunted to 4 V.

Closing SW1 or SW2 forms a divider network that allows the computer to read these higher voltages. A 10:1 divider is formed by closing SW1. The result is a divider network consisting of the 1 M Ω resistor R1, and a 111 k Ω resistor R2 to ground. An 8 V input signal would be read as 0.800 V at the input of IC 1. The programmer should keep in mind that a divider was used on that channel and multiply the answer by 10 when recording it.

Closing SW 2 forms a 100:1 divider. The mathematics is the same except that the resistor (R3) is now 11.11 k Ω . An 8 V input would become 0.080 V and a 150 V input would become 1.500 V. Obviously, proper range selection is necessary to maximize resolution.

An additional feature of this interface is the ability to accommodate AC inputs. This is accomplished by simply converting the AC signal to DC after the divider section output. IC 6 and IC 7 function as single-pole, double-throw switches to gate the converter in or out of the signal path. The actual AC-to-DC converter is shown in figure 8.18.

This device is known as an average RMS (Root Mean Square) converter. If you apply a 1.0 V peak AC signal to it, it will output 0.707 VDC. This is the technique used in most digital multimeters. This is also the way we commonly express AC voltages. For example, household 115 VAC is 115 V average RMS. The peak is about 176 V. The converter passes both AC and DC because there is no blocking capacitor on the input. If it is inadvertently switched into a DC signal, it will multiply the reading by 1.414.

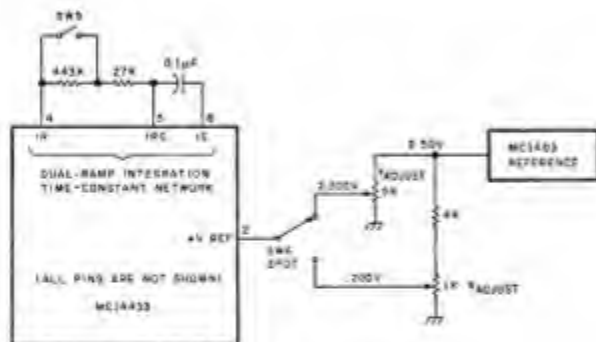


Figure 8.16 Voltage reference and integration time-constant modification circuitry for the digital voltmeter.

Exercising the Interface with a Software Driver

The interface is attached to ZAP through I/O ports. It takes 10 input bits and 8 output bits for full operation. They are arbitrarily chosen as ports 1 and 4 for this description. The actual choice will depend on what addresses you wire when you are configuring ZAP. These ports are not used for anything in the original description and will require the proper port hardware to be added. Summarizing the I/O requirements for the DVM (digital voltmeter) interface:

Command Output Byte (port 1 output)

B7	EOC enable or disable	Disable=1; Enable=0
B6	AC or DC select	AC=0; DC=1
B5	2.0 V or 0.2 V range	2.0 V=0; 0.2 V=1
B4	gain code	0,0=X1
B3		0,1=X10
B2		1,0=X100
B2	channel code	channels 0-7 binary
B1		
B0		

Status Input Byte (port 4 input)

B7	not used
B6	
B5	
B4	
B3	
B2	out of range
B1	
B0	
B0	end of conversion

Data Input Byte (port 1 input)

B7	1st digit	digit enable
B6	2nd digit	
B5	3rd digit	
B4	4th digit	
B3	BCD value	
B2		
B1		
B0		

when B7=0 then: B6

B5	not used
B4	
B3	1/2 digit value
B2	polarity
B1	not used
B0	autoranging status bit

This interface uses a software driver to reduce hardware complexity. The program is not unlike a communications driver. To obtain data from the interface effectively, the computer must be synchronized with the DVM chip and must perform a specific sequence of operations to demultiplex the input data stream.

The actual program that interfaces to and stores the values from the DVM chip is written as a subroutine. All the information necessary for proper execution of the driver is provided in the DE register pair at the time of the call. Its contents will tell the interface which channel to set, whether it should be AC or DC, and which V_{REF} and gain to use. One channel is converted every time the driver routine is called.

The information set in the DE register pair at the time of the call is the command out-

put byte (port 1 output), and each bit has the designations previously listed. The only difference is that bit 7 (the enable/disable bit to the A/D converter) is sent as a logic 0 when doing a call. The driver will set it to an enable condition after it has pulled in the proper relay and allowed a 1.3 ms bounce delay.

Demultiplexing the output of the DVM chip is fairly straightforward. Following the call, the outputs to the interface close the proper switches, and the central processor hangs in a loop waiting for an end-of-conversion signal. When this happens, the program knows that the next 4 digits of data are what it wants. The DVM chip sets each of the digit select lines successively, and the program records the values of the 4 BCD data lines each time. It strips the status and polarity bits from the MSD $\frac{1}{2}$ -digit byte and reformats and stores the voltage input value in 4 bytes of memory. The 3 whole digits are stored in BCD notation and occupy 3 of the bytes. The $\frac{1}{2}$ digit, polarity, and out of range indication are located in the fourth byte. Polarity is indicated by setting the MSB. A positive reading is a logic 1 and a negative input is a logic 0. The $\frac{1}{2}$ -digit value can only be a 0 or 1 and occupies the LSB of the quantity. Out of range is handled with a little program manipulation. If the driver detects that the incoming reading is not within range, it sets the equivalent of +2 in the $\frac{1}{2}$ -digit byte. Obviously, this is an illegal condition for a DVM only capable of counting to 1999. The programmer using this stored data should check the limits of the data before acting upon it.

When the driver completes its operation, it has acquired a $3\frac{1}{2}$ -digit reading and stored it as 4 bytes in a special table in memory. The 8 channels of data constitute a 32-byte table. The location of a particular channel's data is found by a simple expression:

$$\text{The 4-byte data starts at memory location } L + 4(N - 1)$$

where L = starting address of memory table
 N = channel number (1 to 8)

Figure 8.19 is the assembly listing of the program that exercises this DVM interface. When assembled, it occupies less than a page of memory.

Note: One caution should be kept in mind when measuring AC signals with this interface. The ground on the DVM interface is the same as the computer's and a potential short circuit exists unless either the computer power supply or the measured voltage is isolated.

```

0100 *
0110 *** MC14433 3 1/2 DIGIT A/D CONVERTER DRIVER
0120 *
0125 * REV. 1.9
0130 *
0140 DIP      EQU 1      DATA INPUT PORT NUMBER
0150 SIP      EQU 4      STATUS INPUT PORT NUMBER
0160 COP      EQU 1      COMMAND OUTPUT PORT NUMBER
0170 EEOC     EQU 200    ENABLE EOC INPUT
0180 DEOC     EQU 000    DISABLE EOC INPUT
0190 *
0200 *
0210 * CONVERTED CHANNEL DATA BUFFERS
0220 *
0230 CHAN0    DW 000000
0240         DW 000000
0250 CHAN1    DW 000000
0260         DW 000000
0270 CHAN2    DW 000000
0280         DW 000000
0290 CHAN3    DW 000000
0300         DW 000000
0310 CHAN4    DW 000000
0320         DW 000000
0330 CHAN5    DW 000000
0340         DW 000000

```

Figure 8.19 A listing of the assembly-language program that exercises the digital voltmeter.

```

0350 CHAN6 DW 000000
0360 DW 000000
0370 CHAN7 DW 000000
0380 DW 000000
0390 *
0400 * INTERMEDIATE DATA BUFFERS
0410 *
0430 CHAN DB 000 CURRENT CHANNEL NUMBER
0440 CCP DW 000000 COMMAND CHANNEL PARAMETER
0460 *
0470 *
0480 *** START A/D CONVERTER
0490 *
0550 *
0560 START LD A,E
0570 LD (CCP),A
0580 AND 007
0590 LD (CHAN),A
0600 LD IX,CHAN0
0910 LD D,0
0920 LD E,A
0930 SLA E CALCULATE BUFFER OFFSET
0940 SLA E
0950 ADD IX,DE
0960 *
0970 * SELECT CHANNEL AND START CONVERSION
0980 *
0985 LD B,3 SET CYCLE COUNT
0990 SCSC LD A,(CCP)
1000 OUT COP SELECT CHANNEL
1005 CALL DELAY
1010 OR EEDC ENABLE EOC OUTPUT
1020 OUT COP COMMAND A/D CONVERTER
1030 *
1040 * WAIT FOR EOC
1050 *
1060 WEOC IN SIP READ CONVERTER STATUS
1070 BIT 0,A TEST FOR EOC
1080 JR Z,WEOC JUMP IF NOT READY
1085 DJNZ SCSC
1090 BIT 1,A TEST FOR OVERANGE
1100 JR NZ,OVER JUMP IF TRUE
1110 *
1120 * CONVERSION DONE#PROCESS FIRST (MSD) DIGIT
1130 *
1140 MSD0 LD B,200 SELECT DIGIT 1
1150 CALL RDIG WAIT AND READ DIGIT 1
1160 CPL
1170 RRCA RIGHT JUSTIFY DIGIT VALUE
1180 RRCA
1190 RRCA
1200 AND 1 ISOLATE
1210 LD E,0 INITIALIZE STATUS BYTE
1220 BIT 2,D TEST POLARITY
1230 JR NZ,MSD3 JUMP IF POSITIVE
1240 LD E,200 LOAD POLARITY SIGN
1440 *
1450 * SAVE MSD AND CURRENT POLARITY
1460 *
1470 MSD3 OR E ADD POLARITY SIGN TO MSD
1480 LD (IX+0),A SAVE IN DATA BUFFER
1500 *
1510 * PROCESS 2ND DIGIT
1520 *
1530 RRC B SELECT DIGIT 2

```

Figure 8.19 continued

```

1540      CALL RDIG  WAIT AND READ DIGIT
1550      AND 017   ISOLATE
1560      LD  (IX+1),A STORE SECOND DIGIT
1570 *
1580 * PROCESS 3RD DIGIT
1590 *
1600      RRC 8      SELECT 3RD DIGIT
1610      CALL RDIG  WAIT AND READ DIGIT
1620      AND 017   ISOLATE
1630      LD  (IX+2),A STORE
1640 *
1650 * PROCESS 4TH DIGIT
1660 *
1670      RRC 8      SELECT 4TH DIGIT
1680      CALL RDIG  WAIT AND READ DIGIT
1690      AND 017   ISOLATE
1700      LD  (IX+3),A STORE
1710 RAPUP RET
1720 *
1730 * LOAD 2,000 OVERRANGE VALUE INTO DATA BUFFER
1740 *
1750 OVER  LD  A,2    LOAD MSD VALUE
1760      LD  (IX+0),A
1770      XOR A
1780      LD  (IX+1),A LOAD LSD VALUES
1790      LD  (IX+2),A
1800      LD  (IX+3),A
1810      JR  RAPUP
1870 *
1880 *
1890 * READ DIGIT ROUTINE
1900 *
1910 RDIG  IN  DIP    READ DATA BYTE
1920      CPL      CONVERT TO HIGH TRUE LOGIC
1930      LD  D,A    SAVE COPY
1940      AND 8      TEST FOR GIVEN DIGIT READY
1950      JR  Z,RDIG JUMP IF NOT
1960      LD  A,D    RESTORE A REGISTER
1970      RET      RETURN TO CALLER
1980 DELAY LD  C,377
1990 DELI  DEC C
2000      RET Z
2010      JR  DELI

```

Figure 8.19 continued

Potential Applications

I feel that data acquisition is a natural application for ZAP. The interface outlined above can be used in a solar heating system to monitor and record pertinent data. Using the facilities of the ZAP monitor and the DVM interface routine, an 8-channel data logger is practical. In general, all that would be required is a supervisory program that calls the DVM 8 times to obtain the 8 sensor inputs. It then sets the limits of the memory table to a serial output subroutine and stores the readings on a cassette. This could be done continuously or at regular intervals. The ultimate system would include a real-time clock so that these readings, as well as the times at which they were taken, could be recorded.

Real-Time Clock

If ZAP is going to be used for critical data acquisition or control functions, consideration should be given to real-time synchronization with process events. A simple definition of a real-time system is one that responds to the need for action in a period of

time proportional to the urgency of the need. It boils down to the fact that the computer must be capable of performing a specific action at a specific time. For this to happen, the computer must be able to "tell time."

We can accomplish this by using either software or hardware applications. The simplest technique is to use a clock circuit (figure 8.20) to provide a time tick to the central processor's nonmaskable interrupt line. It can be every 60th, 10th, or 1 second, as suggested in the schematic. When the computer acknowledges the interrupt, it first saves all the registers from the program it was executing, and then services the real-time interrupt. Frequently, the first action is to increment an internal counter that keeps track of elapsed time. Usually it's a value equivalent to the total number of clock ticks, whether in seconds or milliseconds. Once this regular interval has been established, it is easy for the computer to perform real-time functions.

Clock resolutions down to milliseconds sound great and make interval timing extremely accurate. However, I doubt most ZAP builders would want to use such an interface in light of the complex software involved. I much prefer an interface that is easier to implement and more likely to be used.

Essentially, the kind of real-time system most appealing to ZAP owners has a resolution of perhaps 1 minute rather than 1 ms. Also, it's best if it can be read directly in hours and minutes rather than as a total clock count. A direct benefit is reduced overhead. The computer does not have to acknowledge the clock update or scan status flags as often. At first glance, it may not seem like much of a saving, but some routines can use up to 10 percent of the processor time handling a millisecond clock interrupt.

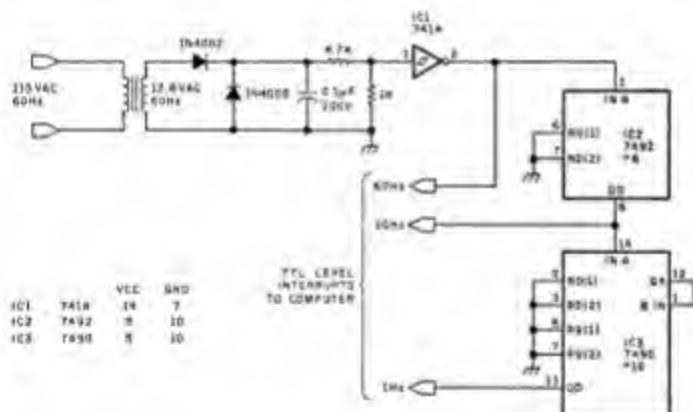


Figure 8.20 A simple time-base generator for an interrupt-driven real-time clock.

An Old Clock Chip to the Rescue

The easiest way to provide an hourly and minute-by-minute input is to interface the computer to a MOS/LSI clock chip similar to that found on most digital clocks or watches. There are two approaches to the design of a clock interface: one method is to let the clock circuit operate independently from the computer, attached in such a way that the computer can monitor the output lines and extract a time value on the fly. The software necessary for this approach would be very much like the DVM interface described previously. The other method, which I prefer because it involves less software, is to give the computer complete control over the information flow of the clock in a synchronous manner.

Figure 8.21 shows such a clock interface. This circuit, manually preset to keep it simple, is computer directed. The basic 4-chip circuit consists of an MM5312 4-digit BCD/7-segment output digital clock chip, an MM5369 time-base generator, and two MOS-to-TTL buffers to send data to the processor.

Time is set on the chip by grounding the slow and fast set lines, pins 14 and 15. To know what is being set you must read the interface at the same time, and display the time on the 4-digit hexadecimal address display, already part of the expanded ZAP. Time is read from the interface as 4 binary-coded decimal numbers. The 8 input lines to the computer are attached to an 8-bit parallel input port, and are divided between 4 digit-enable lines, and 4 BCD digit-value lines. Data appear as a digit enable and an associated BCD number. The tens of minutes data is read on B0 thru B3 when B5 is high (B4, B6, and B7 are low). Similarly, B0 thru B3 will hold the tens of hours quantity when B7 is high. The interface logic will stay on a particular digit until it is instructed to proceed to the next digit. Sequencing is under program control and uses one output bit of a convenient parallel port.

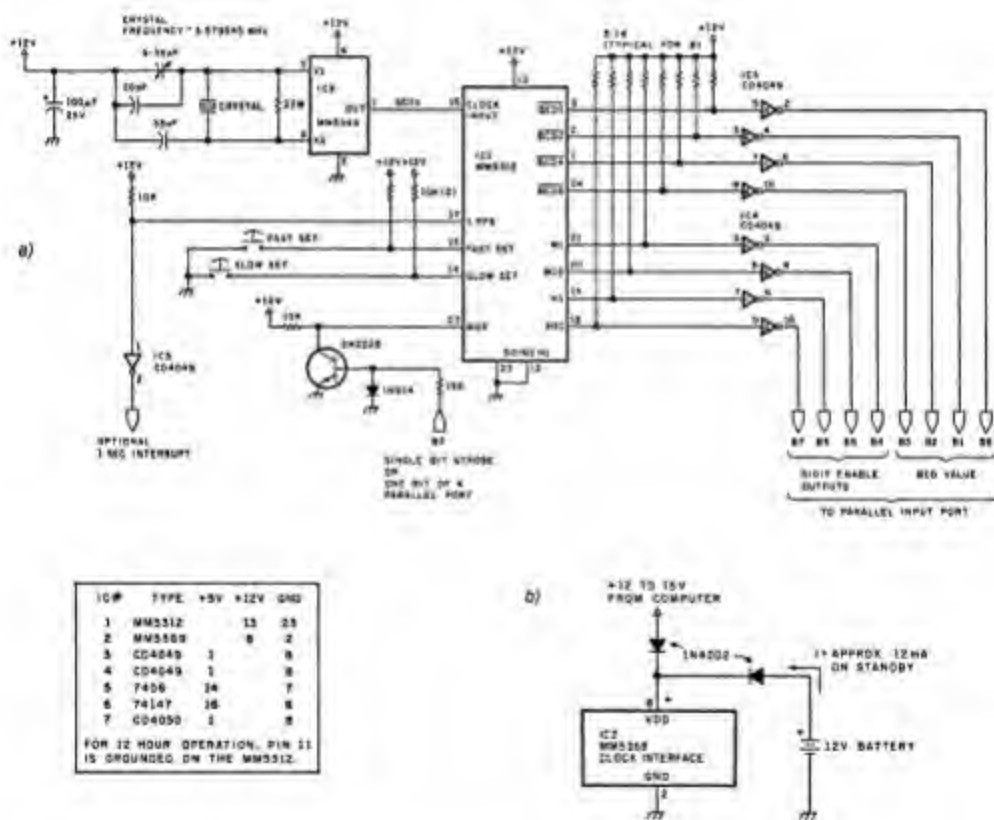


Figure 8.21 A schematic diagram of a real-time clock interface.

- Using a MOS digital clock chip.
- With battery backup.

Figure 8.22 shows how the multiplexer line is controlled in this application. One bit of an output port is used to pulse multiplexer input pin 22. (All that is required is a 1 ms pulse. As an alternative, a one-shot could be triggered from a decoded strobe line of an unwired port.) At any time, 1 of the 4 digit-enable lines will be low and a digit's value will be on the BCD output lines. Just determine which digit it is and store the value. Next we pulse the multiplexer input to enable the next digit and save it as well. Conceivably, it takes only 4 iterations of this procedure to obtain a complete 4-digit reading. If you prefer a more orderly approach, you can follow the program flow outlined in figure 8.23. The only difference is that it waits until the chip cycles to the beginning before storing the readings.

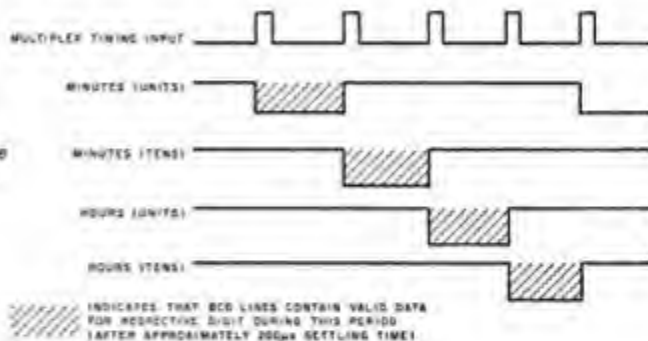


Figure 8.22 The multiplex timing sequence for the display in the circuit of figure 8.21.

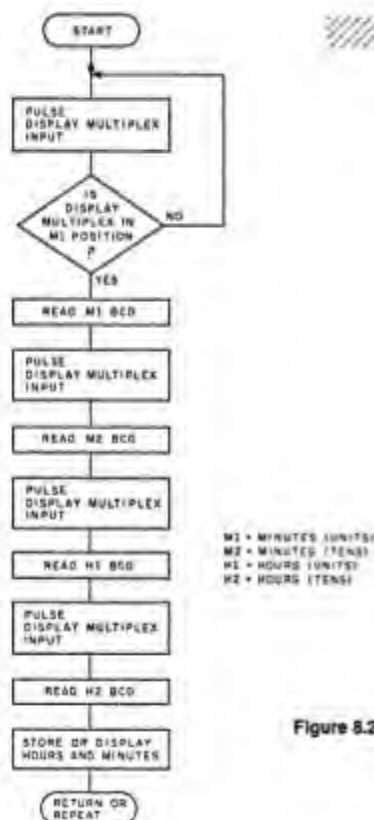


Figure 8.23 A flowchart of a program for the circuit in figure 8.21.

CHAPTER 9

BUILD A CRT TERMINAL

LOW COST VERSATILE CRT TERMINAL

This chapter describes the design of a low-cost features-oriented cathode-ray tube (CRT) terminal. Two MOS/LSI devices from Standard Microsystems Corporation reduce the number of parts required for a CRT terminal yet enhance its capabilities.

The two devices, the CRT 5027 video timer and controller and the CRT 8002 video display attributes controller, provide virtually all of the circuitry for the display portion of the CRT terminal. (See Appendices C8 and C9 for specifications.)

The terminal is designed to stand alone and communicate via an RS-232C interface with any computer system. If, in the expanded ZAP, the 6-character hexadecimal display proves inadequate, then the experimenter has only to construct this unit and attach it to the serial port already assembled.

Device Description

The CRT 5027 contains the logic required to generate all of the timing signals (vertical and horizontal synchronization, page refresh memory address, etc.) required by a CRT terminal. The entire display format including interlace/non-interlace, characters per row, rows per frame, scans per row, horizontal synchronization pulse width, and timing are user programmable for all standard and most nonstandard formats.

Although the CRT 5027 is basically structured for use with its own microprocessor, this design describes a "dumb terminal" using a low-cost PROM and standard TTL logic to replace the microprocessor control. While increasing the number of the parts, this design results in a low-cost, high quality alphanumeric/graphics terminal.

The CRT 8002 provides a 7×11 dot matrix, 128 character generator ROM, and a high-speed video shift register cursor. It includes logic to generate such functions as underline, blinking, reverse video, blanking, and strike-through. Additional wide and thin graphics modes allow the creation of line drawings, forms and unique graphic symbols.

Terminal Description

As with most electronic designs, a CRT terminal involves a large number of performance and cost trade-offs. A screen format of 16 rows of 64 characters per row was selected to minimize memory requirements (1 K bytes) and keep the video frequency within the limits of lower cost video monitors. An 80-character line would have not only increased the video frequency beyond the bandwidth of many low-cost monitors, but also would have increased the memory requirements. Similarly, more rows per page would have increased the memory requirement unless the characters per line were reduced.

In many microprocessor applications, the page memory is shared with the processor via a data bus. In this application, the page memory is used strictly by the CRT with data input synchronously, character-by-character, into the cursor position.

Full graphics or attributes may be selected on a character-by-character basis using

control words on the input data bus. A block diagram of the terminal is shown in figure 9.1.

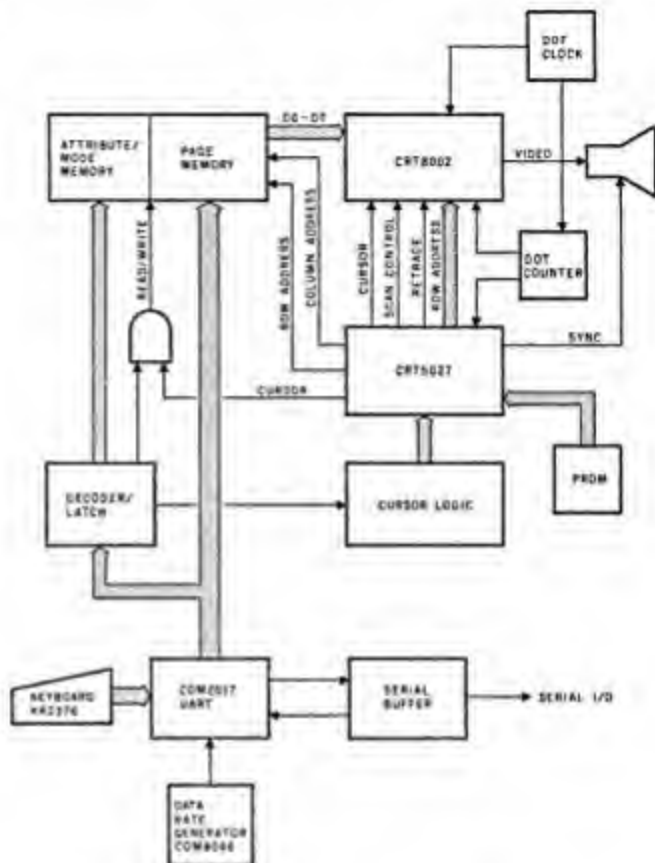


Figure 9.1 A block diagram of a low-cost cathode-ray tube terminal.

Character Format

The CRT 8002 requires a minimum 8×12 character block to form its basic 7×11 character and to provide line and character spacing. However, in order to allow framing a character fully for a reverse video presentation, the horizontal character block must be increased to 9 or 10 dots. For the same reason, allocating 13 lines per character allows top and bottom framing as well.

With the standard TV sweep rates of 60 Hz (vertical) and 15,750 Hz (horizontal), there are $15,750 \div 60 = 262.5$ lines per frame. As non-interlaced operation requires an even number of lines, a horizontal frequency of 15,720 Hz is used. The 16 rows \times 13 scan lines per row result in 208 lines of displayed data. The remaining 54 lines will be automatically blanked by the CRT 5027 and will provide upper and lower margins.

To allow for left and right margins as well as for retrace time, a total 80 character times are allocated per line. A good rule of thumb is that the total number of character

times is 25% greater than the actual number of displayed characters.

The video clock frequency is calculated as follows: $10 \text{ (dots per character)} \times 80 \text{ (character times per line)} \times 15,720 \text{ Hz (horizontal sweep frequency)} = 12.576 \text{ MHz}$. See the worksheet in table 9.1.

1. H CHARACTER MATRIX (No. of Dots):	7
2. V CHARACTER MATRIX (No. of Horiz. Scan Lines):	11
3. H CHARACTER BLOCK (Step 1 + Desired Horiz. Spacing = No. in Dots):	10
4. V CHARACTER BLOCK (Step 2 + Desired Vertical Spacing = No. in Horiz. Scan Lines):	13
5. VERTICAL FRAME (REFRESH) RATE (Freq. in Hz):	60
6. DESIRED NO. OF DATA ROWS:	16
7. TOTAL NO. OF ACTIVE "VIDEO DISPLAY" SCAN LINES (Step 4 x Step 6 = No. in Horiz. Scan Lines):	208
8. VERT. SYNC DELAY (No. in Horiz. Scan Lines):	26
9. VERT. SYNC (No. in Horiz. Scan Lines; $T = 190 \mu\text{s}^*$):	3
10. VERT. SCAN DELAY (No. in Horiz. Scan Lines; $T = 1.59 \mu\text{s}^*$):	25
11. TOTAL VERTICAL FRAME (Add steps 7 thru 10 = No. in Horiz. Scan Lines):	263
12. HORIZONTAL SCAN LINE RATE (Step 5 x Step 11 = Freq. in KHz):	15.720
13. DESIRED NO. OF CHARACTERS PER HORIZ. ROW:	64
14. HORIZ. SYNC DELAY (No. in Character Time Units; $T = 4.77 \mu\text{s}^{**}$):	6
15. HORIZ. SYNC (No. in Character Time Units; $T = 5.57 \mu\text{s}^{**}$):	7
16. HORIZ. SCAN DELAY (No. in Character Time Units; $T = 2.38 \mu\text{s}^{**}$):	3
17. TOTAL CHARACTER TIME UNITS IN (1) HORIZ. SCAN LINE (Add Steps 13 thru 16):	80
18. CHARACTER RATE (Step 12 x Step 17 = Freq. in MHz):	12.576
19. CLOCK (DOT) RATE (Step 3 x Step 18 = Freq. in MHz):	12.576

*Vertical interval
**Horizontal interval

Table 9.1 A CRT 5027 worksheet for a 64 characters per row, 16 row, noninterlaced screen format.

Programming the VTAC

The CRT 5027 VTAC (Video Timer and Controller) is user programmable for all timing and format requirements. The programming data is stored in 9 on-chip registers. Although a microprocessor can easily provide the programming data, a low-cost PROM is used in this application. The 9 registers are programmed as follows (see table 9.2):

Register 0: This register contains the number of character times for one horizontal period, and is normally 1.25 times the number of characters per line, in this case $64 \times 1.25 = 80$. As the internal counters are initialized at zero, the actual number in the register is $80 - 1 = 79$.

0	1	0	0	1	1	1	1
---	---	---	---	---	---	---	---

Register 0

Register 1: This has 3 fields:

- 1) bit 7 — one for interlace, zero for non-interlace. In this example, noninterlaced operation is selected.
- 2) bits 3 thru 6 program the number of character times for the width of the horizontal synchronization pulse. This parameter is monitor dependent and is typically

5 μ s. Because there are 80 character times for a 63.6 μ s horizontal scan time ($1 \div 15,720$), each character time is 0.801 μ s; 7 character times will be used to generate a 5.56 μ s pulse.

- 3) bits 0 thru 2 set the horizontal "front porch." This essentially positions the data horizontally. The monitor's specification will determine initial programming although some experimentation may be required to center the display exactly. Six character times were selected for the front porch.

0 0 1 1 1 1 1 0

Register 1

REG. #	ADDRESS A3 A0	FUNCTION	BIT ASSIGNMENT	HEX	DEC.
0	0000	HORIZ. LINE COUNT <u>80</u>	0 1 0 0 1 1 1 1	4F	79
1	0001	INTERLACE <u>0</u> H SYNC WIDTH <u>9</u> H SYNC DELAY <u>6</u>	0 0 1 1 1 1 1 0	3E	62
2	0010	SCANS/DATA ROW <u>13</u> CHARACTERS/ROW <u>64</u>	X 1 1 0 0 0 1 1	63	99
3	0011	SKW. CHARACTERS <u>1</u> DATA ROWS <u>16</u>	1 0 0 0 1 1 1 1	8F	143
4	0100	SCANS/FRAME <u>262</u> $8 = \frac{262}{32}$	0 0 0 0 0 0 1 1	03	3
5	0101	VERTICAL DATA START $= 3 + \text{VERTICAL SCAN DELAY}$ SCAN DELAY <u>35</u> DATA START <u>38</u>	0 0 0 1 1 1 0 0	1C	28
6	0110	LAST DISPLAYED DATA ROW (= DATA ROWS)	X X 0 0 1 1 1 1	0F	15

Table 9.2 A CRT 5027 register-programming worksheet for a 16 x 64 screen format.

Register 2: This has two fields:

- bits 3 thru 6 (bit 7 is not used) set the number of scans per character. In this case, we have defined the character as 10×13 , so the binary equivalent of $13 - 1 = 12$ is used (all CRT 5027 counters start at zero, not one, so programming of counters is always one less than the number).
- bits 0 thru 2 contain a 3-bit code for the number of characters per line. From the data sheet the code for 64 is 011.

0 1 1 0 0 0 1 1

Register 2

Register 3: This has two fields:

- bits 6 and 7 delay the blanking cursor and synchronization timing to allow for character generator and programmable memory propagation delays. Generally, one character time will allow for these delays.
- bits 0 thru 5 define the number of data rows, once again starting with binary zero for one line. $16 - 1 = 15$ will be programmed.

1 0 0 0 1 1 1 1

Register 3

Register 4: Register 4 sets the number of raster lines per frame. For the noninterlaced mode this is derived by the formula $(N - 256) \div 2 = 3$.

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Register 4

Register 5: This contains the number of raster lines between the start of the vertical synchronization pulse and the start of data (vertical synchronization + back porch). This time must be long enough to allow for the full retrace time of the monitor and to allow vertical positioning of the display. We will use 28 here. The front porch will be calculated by the CRT 5027 as $262 - (13 \times 16) - 28 = 26$.

0	0	0	1	1	1	0	0
---	---	---	---	---	---	---	---

Register 5

Register 6: Register 6, the scrolling register, is programmed with the number of the last data row to be displayed. Since we want to initialize the CRT 5027, this will be programmed the same as Register 3 (bits 6 and 7 are not used).

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

Register 6

Register 7 and Register 8: These registers contain the cursor character number and row number respectively. Since the cursor is to be initially positioned at the top left corner, both registers will be initialized with all zeros. Subsequent cursor position changes will be entered as described under "circuit operation."

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Register 7

Register 8

Circuit Description

Referring to figure 9.2, IC 1A, IC 1B, IC 4 provide the video dot clock (12.58 MHz) and the character clock DCC, which is the dot clock $\div 10$ (each character is 10 dots wide). The video dot clock determines the actual video data rate. The character clock determines the speed each character is addressed. IC 6A buffers the dot clock input of the CRT 8002. A pull-up resistor is used on the output to guarantee the logic one requirement of the VDC input.

The LOAD command loads the register information required for programming the CRT 5027 from the PROM IC 7 to the CRT 5027. The "self-load" capability of the CRT 5027 is used to automatically scan the PROM addresses. LOAD is automatically generated on power-on by IC 1D.

Because of the bus structure of the CRT 5027, cursor position information is loaded on the same bus as the register data. Three-state data selectors IC 14 and IC 15 select cursor X position data from counter IC 8 and IC 7 or cursor Y position data from IC 1D. IC 12 and IC 13 select the address mode for the CRT 5027. Three modes are used: "nonprocessor self-load" for register loading, load cursor X position, and load cursor Y position.

IC 16 thru IC 21 decode attribute mode and cursor controls from the ASCII data bus. If graphics or special attributes are not desired, IC 16, 17, and 21 are not required. Similarly, if cursor controls are directly available, decoding them is not necessary.

IC 19 and IC 20 are 256×4 PROMs. Their exact programming can be suited to the user needs. The programming used in this terminal is shown in table 9.3. When a key designated as an attribute or mode key is depressed, the appropriate control word is latched in IC 21; all subsequent data entries will have that word loaded in the upper 4 bits of programmable memory. This allows the attribute or mode to be changed on a character-by-character basis. IC 18, a 2 to 4 decoder, is enabled when a cursor control backspace, carriage return/line feed, or 1 is decoded and provides the appropriate cursor movement.

TTL or low power TTL can be used throughout. Schottky TTL is recommended for IC 6 due to the fast rise time requirements of the clock input.

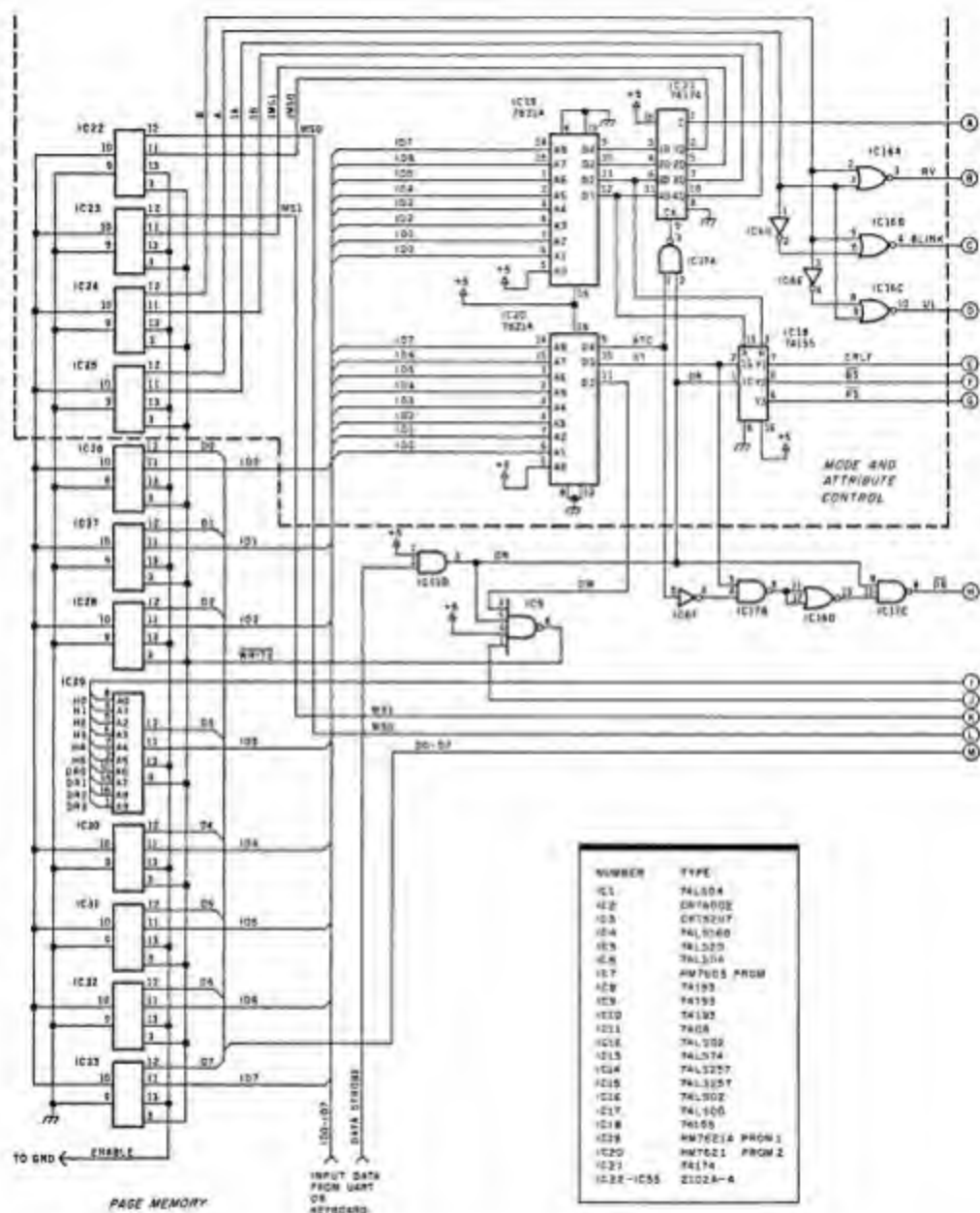


Figure 9.2 A schematic diagram of a low-cost versatile CRT terminal using the CRT 5027 and CRT 8002 chips (continued on next page).

Operation

After power-on, Control Q should be depressed to latch the system in the "normal" mode. Depressing the space key and the erase key simultaneously will then blank the screen. All further character entries will be displayed normally. If other attributes or graphics are desired, the appropriate control code is entered. This character will not be displayed or cause cursor movement, but will latch the new command. Modes may be changed for every character desired. Cursor movement may be decoded from the ASCII input by the control key as indicated in table 9.3.

PROM Programming

Keyboard Entry	Function	Address	PROM 1 Output	PROM 2 Output
		76543210	D ₇ D ₆ D ₅ D ₄	D ₇ D ₆ D ₅ D ₄
Return	Carriage Return	00011011	0011	1000
LF	Line Feed	00010101	1011	1000
Control H	Cursor Left	00010001	0111	1000
RS	Cursor Up	00111101	1111	1000
US	Cursor Right	00111111	1111	1010
Control Q	Normal Attribute	00100011	1111	1011
Control W	Blink	00101111	1011	1011
Control E	Underline	00001011	0111	1011
Control R	Reverse Video	00100101	0011	1011
Control T	External Mode	00101001	1101	1011
Control Y	Wide Graphics	00110011	1100	1011
Control U	Thin Graphics	00101011	1110	1011
Balance of PROM			0011	1110

Table 9.3 PROM programming for the circuit of figure 9.2

The Rest of the System

Figure 9.3 illustrates the balance of the circuitry required to implement a full RS-232C compatible serial I/O terminal. Utilization of MOS/LSI reduces the package count to a bare minimum.

A KR2376 keyboard encoder, IC 1, encodes and de-bounces the keyboard switches and provides an ASCII data word to the COM 2017 UART (see Appendices C6 and C7). The UART, in turn, provides the serial receive/transmit interface. The data rate is programmable by means of the switch controlled input code to a COM 8046 data rate generator (see Appendix C10).

TERMINAL VARIATIONS

The terminal described can easily be modified for a wide variety of other screen formats. The following changes are required for an 80-characters per row, 24-row format:

1. Horizontal sweep rate — to allow for the increased number of displayed lines (312), the horizontal sweep rate is increased to 20,220 Hz.
2. The video oscillator frequency is calculated as $9 \text{ (dots per character)} \times 100 \text{ (character times per row)} \times 20,220 = 18.198 \text{ MHz}$. Notice that 9 dots per character was selected instead of 10, as 10 would have resulted in a clock frequency of 20.2 MHz, which is beyond the CRT 8002A's top frequency. IC 4, therefore, must be set for divide by 9 rather than 10.
3. An additional 1 K bytes of page memory is required. Figure 9.4 shows the revised address connections.
4. Register programming for the CRT 5027 follows the worksheet shown in tables 9.4 and 9.5.

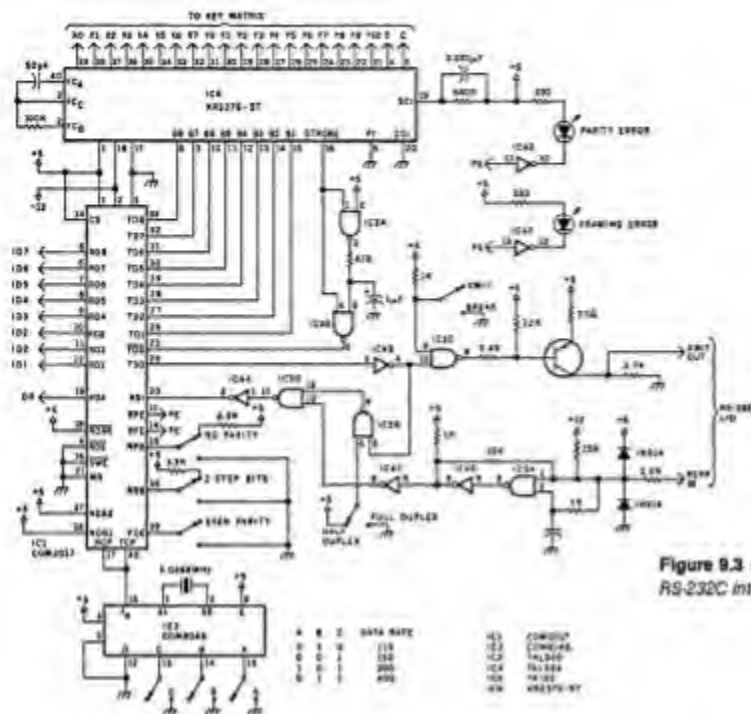


Figure 9.3 A schematic diagram of a RS-232C interface for a terminal.

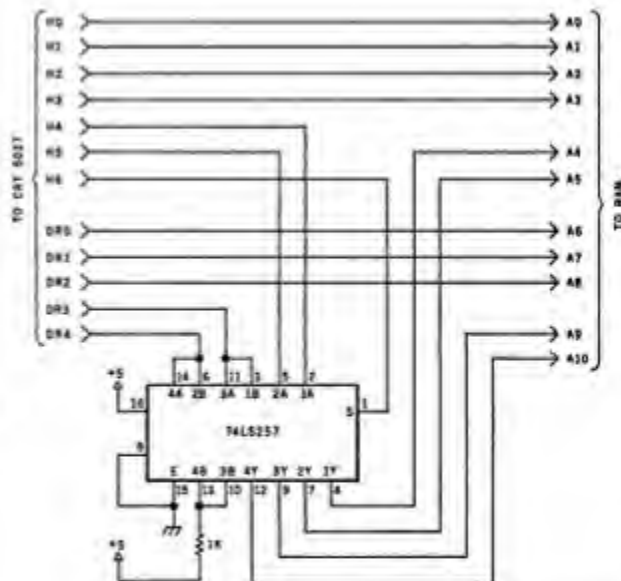


Figure 9.4 A memory-mapping system for a 24 × 80 screen format.

1. H CHARACTER MATRIX (No. of Dots):	<u>7</u>
2. V CHARACTER MATRIX (No. of Horiz. Scan Lines):	<u>11</u>
3. H CHARACTER BLOCK (Step 1 + Desired Horiz. Spacing = No. in Dots):	<u>9</u>
4. V CHARACTER BLOCK (Step 2 + Desired Vertical Spacing = No. in Horiz. Scan Lines):	<u>13</u>
5. VERTICAL FRAME (REFRESH) RATE (Freq. in Hz):	<u>60</u>
6. DESIRED NO. OF DATA ROWS:	<u>24</u>
7. TOTAL NO. OF ACTIVE "VIDEO DISPLAY" SCAN LINES (Step 4 x Step 6 = No. in Horiz. Scan Lines):	<u>312</u>
8. VERT. SYNC DELAY (No. in Horiz. Scan Lines):	<u>3</u>
9. VERT. SYNC (No. in Horiz. Scan Lines; $T = \frac{168.8}{60} \mu s$):	<u>9</u>
10. VERT. SCAN DELAY (No. in Horiz. Scan Lines; $T = \frac{890.2}{60} \mu s$):	<u>18</u>
11. TOTAL VERTICAL FRAME (Add steps 7 thru 10 = No. in Horiz. Scan Lines):	<u>336</u>
12. HORIZONTAL SCAN LINE RATE (Step 5 x Step 11 = Freq. in kHz):	<u>20220</u>
13. DESIRED NO. OF CHARACTERS PER HORIZ. ROW:	<u>80</u>
14. HORIZ. SYNC DELAY (No. in Character Time Units; $T = \frac{148}{60} \mu s$):	<u>5</u>
15. HORIZ. SYNC (No. in Character Time Units; $T = \frac{490}{60} \mu s$):	<u>10</u>
16. HORIZ. SCAN DELAY (No. in Character Time Units; $T = \frac{346}{60} \mu s$):	<u>7</u>
17. TOTAL CHARACTER TIME UNITS IN (1) HORIZ. SCAN LINE (Add Steps 12 thru 16):	<u>100</u>
18. CHARACTER RATE (Step 12 x Step 17 = Freq. in kHz):	<u>2022</u>
19. CLOCK (DOT) RATE (Step 3 x Step 18 = Freq. in kHz):	<u>18198</u>

*Vertical Interval
*Horizontal Interval

Table 9.4 A CRT 5027 worksheet for an 80 characters per row, 24 row, noninterlaced screen format.

REG. #	ADDRESS (A1 A0)	FUNCTION	BIT ASSIGNMENT	HEX	DEC.
0	0000	HORIZ. LINE COUNT <u>100</u>	<u>0 1 1 0 0 0 1 1</u>	<u>63</u>	<u>99</u>
1	0001	INTERLACE <u>0</u> H SYNC WIDTH <u>10</u> H SYNC DELAY <u>5</u>	<u>0 1 0 1 0 0 1 1</u>	<u>53</u>	<u>83</u>
2	0010	SCANS/DATA ROW <u>13</u> CHARACTERS/ROW <u>80</u>	<u>X 1 1 0 0 1 0 1</u>	<u>65</u>	<u>101</u>
3	0011	SKEW CHARACTERS <u>2</u> DATA ROWS <u>24</u>	<u>1 0 0 1 0 1 1 1</u>	<u>97</u>	<u>151</u>
4	0100	SCANS/FRAME <u>336</u> X = <u>40</u>	<u>0 0 1 0 1 0 0 0</u>	<u>28</u>	<u>40</u>
5	0101	VERTICAL DATA START = 3 + VERTICAL SCAN DELAY; SCAN DELAY <u>18</u> DATA START <u>21</u>	<u>0 0 0 1 0 1 0 1</u>	<u>15</u>	<u>21</u>
6	0110	LAST DISPLAYED DATA ROW (= DATA ROWS)	<u>X X 0 1 0 1 1 1</u>	<u>17</u>	<u>23</u>

Table 9.5 A CRT 5027 register-programming worksheet for a 24 x 80 screen format.

APPENDICES

Appendix A

Construction Techniques

CONSTRUCTION TIPS

As a result of building a project every month for my "Clarcia's Circuit Cellar" column in *BYTE* magazine and of constructing every circuit in this book, I feel I can speak as an authority on the subject of prototype construction. A prototype is a nice term that describes the one-of-a-kind kluge that you build from a schematic. This is opposed to the kit or semi-assembled project that includes a printed circuit board which only requires plugging in components.

Prototyping a circuit is not easy. There are many dos and don'ts, but successful prototyping is primarily a function of experience. And experience comes only by building something.

The text is purposely laid out with this philosophy in mind. I suggest that you start with the power supply. Not only is the rest of the computer useless without it, but it has built-in protective circuitry that is very forgiving if you make mistakes. Also, by constructing the power supply first, there is less likelihood of destroying the rest of the computer as you are testing the power supply.

In general, the cardinal rule of prototyping is: be neat. The ZAP computer has high frequencies. Wiring should be the shortest distance between two connections. The longer the wire, the more of an antenna it becomes. In extreme cases, the computer can actually cease to function because of induced electrical noise. With the relatively slower digital signals carried by the wiring attached to external input and output ports, the situation is less critical. Short pulses and high-speed data, such as the signals on the central processor control and address lines, are more critical. In these cases, it is always a good idea to use additional protective circuitry such as buffers.

To a certain degree, the ZAP computer can be laid out as you see fit. Figure A.1 suggests one approach: it can be wirewrapped or hand soldered. Almost any board large enough to accommodate all the chips should suffice. A good choice is a standard S-100 prototyping card available at most computer stores. There is no particular bus other than the standard Z80 signals designated for ZAP because it is primarily intended as a single-board system. The 100-pin connector provides a convenient I/O and power connector. Care should be taken if you decide to split the computer schematic and assemble the computer on more than one board. The separation should be between logical subsystems; for maximum success, all signals should be buffered in and out of the board, e.g., all the memory could be put on a separate card. As outlined in the text, the address and data lines necessary to this function are already properly buffered.

The question of wirewrapping versus soldering is the builder's prerogative. Personally, I prefer point-to-point hardwiring because it's easier to modify when troubleshooting. Wirewrapping might be easier where the ZAP circuit has already been tested and refined.

Long power-supply daisy chains should be avoided. Rather than running a single +5 V and ground wire, it is better to use a double-sided prototyping board so that the top and bottom sides of the board can be set to ground and +5 V respectively. With this approach, each chip can be plugged in (using IC sockets) and the power leads soldered directly to the copper planes. Wirewrapping or not, it is a good idea to solder the power leads to reduce the potential of intermittent connections. Using the ground

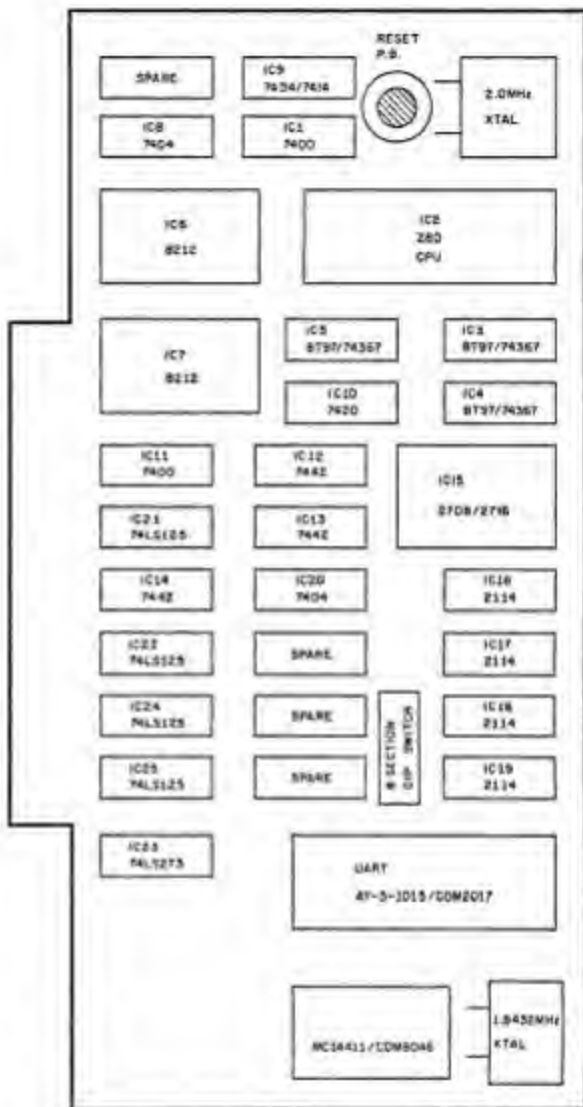


Figure A.1 A typical layout of the basic ZAP computer.

plane for wiring is one of the best ways to reduce noise in computers. If you don't have a ground plane, then solder heavy wire around the perimeter of the circuit board and run short jumpers to it.

Decoupling capacitors are another must for computer prototyping. Digital-integrated circuits, while being virtually burn-out proof in most applications, are unfortunately susceptible to noise carried along the power lines. Often, it will cause them to

go into oscillation. By placing a 0.01 μ F to 0.1 μ F capacitor between +5V and ground about every third IC, the problem is eliminated. Another good idea is to place an electrolytic capacitor at the entrance of any DC power connection to the board. Generally, capacitors are tantalum and three pieces would be required for ZAP's three supplies.

Finally, if you like the concept of ZAP but would rather spend more time applying the finished product than testing your construction techniques, you can look into purchasing EPROMs programmed for the ZAP monitor. The monitor for the ZAP computer is available in a 2708 or single-volt 2716 EPROM for \$25. Please specify the type you want when ordering. These are available from The Micromint, Inc., 917 Midway, Woodmere, NY 11598. Telephone (516) 374-6793.

Appendix B

ASCII Codes

Dec	Octal	Hex	Parity Space or	Character	Control Keybd. Equiv.	Alternate Code Names
000	000	00	Even	NUL	@	NULL, CTRL SHIFT P, TAPE LEADER
001	001	01	Odd	SOH	A	START OF HEADER, SOM
002	002	02	Odd	STX	B	START OF TEXT, EOA
003	003	03	Even	ETX	C	END OF TEXT, EOM
004	004	04	Odd	EOT	D	END OF TRANSMISSION, END
005	005	05	Even	ENQ	E	ENQUIRY, WRU, WHO ARE YOU
006	006	06	Even	ACK	F	ACKNOWLEDGE, RU, ARE YOU
007	007	07	Odd	BEL	G	BELL
008	010	08	Odd	BS	H	BACKSPACE, FE0
009	011	09	Even	HT	I	HORIZONTAL TAB, TAB
010	012	0A	Even	LF	J	LINE FEED, NEW LINE, NL
011	013	0B	Odd	VT	K	VERTICAL TAB, VTAB
012	014	0C	Even	FF	L	FORM FEED, FORM, PAGE
013	015	0D	Odd	CR	M	CARRIAGE RETURN, EOL
014	016	0E	Odd	SO	N	SHIFT OUT, RED SHIFT
015	017	0F	Even	SI	O	SHIFT IN, BLACK SHIFT
016	020	10	Odd	DLE	P	DATA LINK ESCAPE, DC0
017	021	11	Even	DC1	Q	XON, READER ON
018	022	12	Even	DC2	R	TAPE, PUNCH ON
019	023	13	Odd	DC3	S	XOFF, READER OFF
020	024	14	Even	DC4	T	TAPE, PUNCH OFF
021	025	15	Odd	NAK	U	NEGATIVE ACKNOWLEDGE, ERR
022	026	16	Odd	SYN	V	SYNCHRONOUS IDLE, SYNC
023	027	17	Even	ETB	W	END OF TEXT BUFFER, LEM
024	030	18	Even	CAN	X	CANCEL, CANCL
025	031	19	Odd	EM	Y	END OF MEDIUM
026	032	1A	Odd	SUB	Z	SUBSTITUTE
027	033	1B	Even	ESC	[ESCAPE, PREFIX
028	034	1C	Odd	FS	\	FILE SEPARATOR
029	035	1D	Even	GS]	GROUP SEPARATOR
030	036	1E	Even	RS	^	RECORD SEPARATOR
031	037	1F	Odd	US	_	UNIT SEPARATOR
032	040	20	Odd	SP		SPACE, BLANK
033	041	21	Even	!		
034	042	22	Even	"		
035	043	23	Odd	#		
036	044	24	Even	\$		
037	045	25	Odd	%		
038	046	26	Odd	&		
039	047	27	Even	'		APOSTROPHE
040	050	28	Even	(
041	051	29	Odd)		
042	052	2A	Odd	*		
043	053	2B	Even	+		
044	054	2C	Odd	,		COMMA
045	055	2D	Even	-		MINUS
046	056	2E	Even	.		
047	057	2F	Odd	/		

Dec	Octal	Hex	Parity Space or	Character	Control Keybd. Equiv.	Alternate Code Names
048	060	30	Even	0		NUMBER ZERO
049	061	31	Odd	1		NUMBER ONE
050	062	32	Odd	2		
051	063	33	Even	3		
052	064	34	Odd	4		
053	065	35	Even	5		
054	066	36	Even	6		
055	067	37	Odd	7		
056	070	38	Odd	8		
057	071	39	Even	9		
058	072	3A	Even	:		
059	073	3B	Odd	;		
060	074	3C	Even	<		LESS THAN
061	075	3D	Odd	=		
062	076	3E	Odd	>		GREATER THAN
063	077	3F	Even	?		
064	100	40	Odd	@		SHIFT P
065	101	41	Even	A		
066	102	42	Even	B		
067	103	43	Odd	C		
068	104	44	Even	D		
069	105	45	Odd	E		
070	106	46	Odd	F		
071	107	47	Even	G		
072	110	48	Even	H		
073	111	49	Odd	I		LETTER I
074	112	4A	Odd	J		
075	113	4B	Even	K		
076	114	4C	Odd	L		
077	115	4D	Even	M		
078	116	4E	Even	N		
079	117	4F	Odd	O		LETTER O
080	120	50	Even	P		
081	121	51	Odd	Q		
082	122	52	Odd	R		
083	123	53	Even	S		
084	124	54	Odd	T		
085	125	55	Even	U		
086	126	56	Even	V		
087	127	57	Odd	W		
088	130	58	Odd	X		
089	131	59	Even	Y		
090	132	5A	Even	Z		
091	133	5B	Odd	[SHIFT K
092	134	5C	Even	\		SHIFT L
093	135	5D	Odd]		SHIFT M
094	136	5E	Odd	^		!, SHIFT N
095	137	5F	Even	_		~, SHIFT O, UNDERSCORE
096	140	60	Even	`		ACCENT GRAVE
097	141	61	Odd	a		
098	142	62	Odd	b		
099	143	63	Even	c		
100	144	64	Odd	d		
101	145	65	Even	e		
102	146	66	Even	f		
103	147	67	Odd	g		
104	150	68	Odd	h		
105	151	69	Even	i		
106	152	6A	Even	j		
107	153	6B	Odd	k		
108	154	6C	Even	l		
109	155	6D	Odd	m		
110	156	6E	Odd	n		
111	157	6F	Even	o		
112	160	70	Odd	p		
113	161	71	Even	q		

Dec	Octal	Hex	Parity Space or	Character	Control Keybd. Equiv.	Alternate Code Names
114	162	72	Even	r		
115	163	73	Odd	s		
116	164	74	Even	t		
117	165	75	Odd	u		
118	166	76	Odd	v		
119	167	77	Even	w		
120	170	78	Even	x		
121	171	79	Odd	y		
122	172	7A	Odd	z		
123	173	7B	Even	[
124	174	7C	Odd]		VERTICAL SLASH
125	175	7D	Even	~		ALTMODE
126	176	7E	Even	-		(ALTMODE)
127	177	7F	Odd	DEL		DELETE, RUBOUT

Appendix C Manufacturers' Specification Sheets

Appendix C1



2708 8K (1K x 8) UV ERASABLE PROM

	Max. Power	Max. Access
2708	800mW	450ns
2708L	425mW	450ns
2708-1	800mW	250ns
2708-6	800mW	550ns

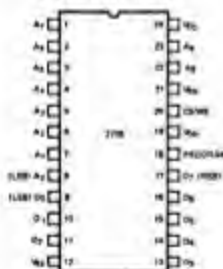
- Low Power Dissipation — 425 mW Max. (2708L)
- Fast Access Time — 350 ns Max. (2708-1)
- Static — No Clocks Required
- Data Inputs and Outputs TTL Compatible during both Read and Program Modes
- Three-State Outputs — OR-Tie Capability

The Intel® 2708 is an 8192-bit ultraviolet light erasable and electrically reprogrammable EPROM, ideally suited where fast turnaround and pattern experimentation are important requirements. All data inputs and outputs are TTL compatible during both the read and program modes. The outputs are three-state, allowing direct interface with common system bus structures.

The 2708L at 425mW is available for systems requiring lower power dissipation than from the 2708. A power dissipation savings of over 50% without any sacrifice in speed is obtained with the 2708L. The 2708L has high input noise immunity and is specified at 10% power supply tolerance. A high-speed 2708-1 is also available at 250ns for microprocessors requiring fast access times.

The 2708 family is fabricated with the N-channel silicon gate FAMOS technology and is available in a 24-pin dual in-line package.

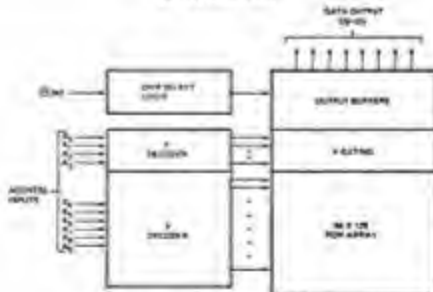
PIN CONFIGURATION



PIN NAMES

A_0-A_{15}	ADDRESS INPUTS
D_0-D_7	DATA OUTPUTS/INPUTS
\overline{OE}	OUTPUT ENABLE (ACTIVE LOW)

BLOCK DIAGRAM



PIN CONNECTION DURING READ OR PROGRAM

		PIN NUMBER									
		DATA HC 8192-BIT		ADDRESS INPUTS 1-8		PROGRAM/ERASE		Vcc		GND	
MODE											
READ	\overline{OE}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6
DATA INPUT	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5
PROGRAM/ERASE	\overline{OE}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6

2708 FAMILY

PROGRAMMING

The programming specifications are described in the Data Catalog PROM/ROM Programming Instructions Section.

Absolute Maximum Ratings*

Temperature Under Bias	-25°C to +85°C
Storage Temperature	-65°C to +125°C
V _{CC} With Respect to V _{SS}	+20V to -0.3V
V _{CC} and V _{AS} With Respect to V _{SS}	+15V to -0.3V
All Input or Output Voltages With Respect to V _{SS} During Read	+15V to -0.3V
CS/WE Input With Respect to V _{SS} During Programming	+20V to -0.3V
Program Input With Respect to V _{SS}	+35V to -0.3V
Power Dissipation	1.5W

*COMMENT

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND A.C. OPERATING CONDITIONS DURING READ

	2708	2708-1	2708-6	2708L
Temperature Range	0°C-70°C	0°C-70°C	0°C-70°C	0°C-70°C
V _{CC} Power Supply	5V ± 5%	5V ± 5%	5V ± 5%	5V ± 10%
V _{DD} Power Supply	12V ± 5%	12V ± 5%	12V ± 5%	12V ± 10%
V _{AS} Power Supply	-5V ± 5%	-5V ± 5%	-5V ± 5%	-5V ± 10%

READ OPERATION

D.C. AND OPERATING CHARACTERISTICS

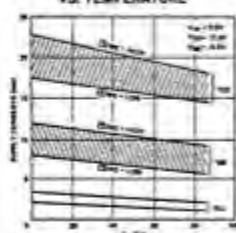
Symbol	Parameter	2708, 2708-1, 2708-6 Limits			2708L Limits			Units	Test Conditions
		Min.	Typ. ¹	Max.	Min.	Typ. ¹	Max.		
I _{CC}	Address and Data Select Input Sink Current	1	10		1	10		μA	V _{AS} = 5.25V or V _{AS} = V _{AS}
I _{OL}	Output Leakage Current	1	10		1	10		μA	V _{CC} = 5.5V, CS/WE = 5V
I _{CC1}	V _{CC} Supply Current	50	60		21	26		mA	Word Case Supply Current ²
I _{CC2}	V _{CC} Supply Current	5	10		2	4		mA	All Inputs High
I _{CC3}	V _{AS} Supply Current	30	40		10	14		mA	CS/WE = 5V, T _A = 0°C
V _{IL}	Input Low Voltage	V _{AS}	0.8V	V _{AS}	0.8V			V	
V _{IH}	Input High Voltage	3.0	V _{CC} - 0.1	2.0	V _{CC} - 0.1			V	
V _{OL}	Output Low Voltage		0.45		0.4			V	V _{CC} = 1.8mA (2708, 2708-1, 2708-6) V _{CC} = 2mA (2708L)
V _{OHS}	Output High Voltage	3.7		3.7				V	V _{CC} = 100μA
V _{OHS}	Output High Voltage	2.4		2.4				V	V _{CC} = 1mA
P _D	Power Dissipation		850		325			mW	T _A = 70°C
					425			mW	T _A = 0°C

NOTES:

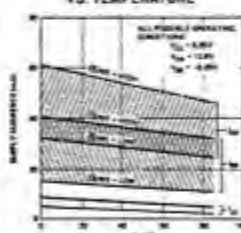
- V_{AS} must be applied prior to V_{CC} and V_{DD}. V_{AS} must also be the last power supply switched off.
- Typical values are for T_A = 25°C and nominal supply voltages.
- The total power dissipation is not calculated by summing the various currents I_{CC}, I_{CC1}, and I_{CC2} multiplied by their respective voltages since current paths exist between the various power supplies and V_{AS}. The I_{CC1}, I_{CC2}, and I_{CC3} currents should be used to determine power supply dissipation only.
- I_{CC3} for the 2708L is specified in the unprogrammed state and is 10mA maximum in the unprogrammed state.

2708 FAMILY

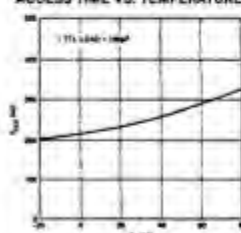
2708L
RANGE OF SUPPLY CURRENTS
VS. TEMPERATURE



2708, 2708-1, AND 2708-6
RANGE OF SUPPLY CURRENTS
VS. TEMPERATURE



ACCESS TIME VS. TEMPERATURE



A.C. CHARACTERISTICS

Symbol	Parameter	2708, 2708L Limits		2708-1 Limits		2708-6 Limits		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{ACC}	Address to Output Delay		450		350		550	ns
t_{CO}	Chip Select to Output Delay		120		120		160	ns
t_{OF}	Chip Deselect to Output Float	0	120	0	120	0	160	ns
t_{OH}	Address to Output Hold	0		0		0		ns

CAPACITANCE⁽¹⁾ $T_A = 25^\circ\text{C}$, $f = 1\text{ MHz}$

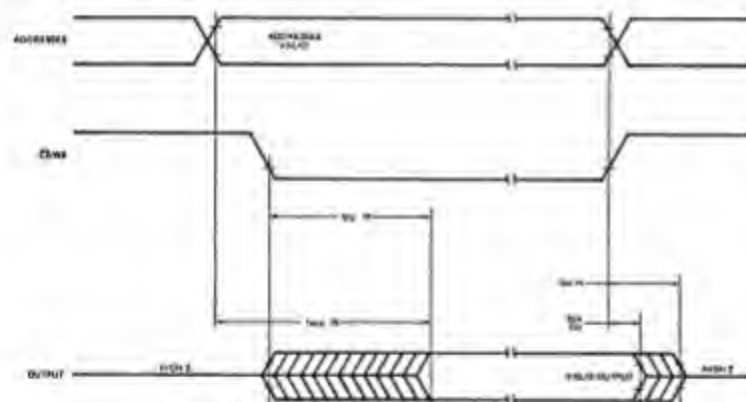
Symbol	Parameter	Typ.	Max.	Units	Conditions
C_{IN}	Input Capacitance	4	6	μF	$V_{DD} = 0\text{V}$
C_{OUT}	Output Capacitance	8	12	μF	$V_{OUT} = 0\text{V}$

NOTE 1: This parameter is periodically sampled and is not 100% tested.

A.C. TEST CONDITIONS:

Output Load: 1 TTL gate and $C_L = 100\text{ pF}$
 Input Rise and Fall Times: 420 ns
 Timing Measurement Reference Levels: 0.5V and 2.8V for inputs; 0.8V and 2.4V for outputs.
 Input Pulse Levels: 0.55V to 3.0V

A.C. WAVEFORMS⁽²⁾



NOTES

2. ALL TIMES SHOWN IN PARENTHESES ARE MINIMUMS AND ARE HOLD TIMES UNLESS OTHERWISE SPECIFIED.
3. CS MAY BE DELAYED UP TO 100 NS AFTER ADDRESSES ARE VALID WITHOUT IMPACT ON t_{ACC} .
4. t_{OH} IS SPECIFIED FROM 25 NS AFTER ADDRESS CHANGES, WHICHEVER OCCURS FIRST.

ERASURE CHARACTERISTICS

The erasure characteristics of the 2708 family are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (\AA). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 \AA range. Data show that constant exposure to room level fluorescent lighting could erase the typical device in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 2708 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 2708 window to prevent unintentional erasure.

The recommended erasure procedure (see Data Catalog PROM/ROM Programming Instructions Section) for the 2708 family is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (\AA). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15 Wsec/cm². The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 $\mu\text{W}/\text{cm}^2$ power rating. The device should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

Appendix C2



2716 16K (2K x 8) UV ERASABLE PROM

- **Fast Access Time**
 - 350 ns Max. 2716-1
 - 390 ns Max. 2716-2
 - 450 ns Max. 2716
 - 490 ns Max. 2716-5
 - 650 ns Max. 2716-6
- **Single +5V Power Supply**
- **Low Power Dissipation**
 - 525 mW Max. Active Power
 - 132 mW Max. Standby Power
- **Pin Compatible to Intel® 2732 EPROM**
- **Simple Programming Requirements**
 - Single Location Programming
 - Programs with One 50 ms Pulse
- **Inputs and Outputs TTL Compatible during Read and Program**
- **Completely Static**

The Intel® 2716 is a 16,384-bit ultraviolet erasable and electrically programmable read-only memory (EPROM). The 2716 operates from a single 5-volt power supply, has a static standby mode, and features fast single address location programming. It makes designing with EPROMs faster, easier and more economical.

The 2716, with its single 5-volt supply and with an access time up to 350 ns, is ideal for use with the newer high performance 48V microprocessors such as Intel's 8086 and 8088. A selected 2716-5 and 2716-6 is suitable for slower speed applications. The 2716 is also the first EPROM with a static standby mode which allows the power dissipation without increasing access time. The maximum active power dissipation is 525 mW while the maximum standby power dissipation is only 132 mW, a 75% savings.

The 2716 has the simplest and fastest method yet devised for programming EPROMs — single byte TTL level programming. No need for high voltage pulsing because all programming controls are handled by TTL signals. Program any location at any time—either individually, sequentially or at random, with the 2716's single address location programming. Total programming time for all 16,384 bits is only 160 seconds.

PIN CONFIGURATION



† Refer to 2732
Data sheet for
pin locations

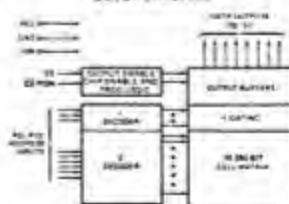
PIN NAMES

Pin	Symbol	Function
1	CE	Chip Enable (Active Low)
2	OE	Output Enable (Active Low)
3	Y ₀ -Y ₇	Data Inputs/Outputs

MODE SELECTION

MODE	CE	OE	Y ₀	Y ₇	OUTPUTS
Read	High	High	High	High	Outputs
Standby	High	High	High	High	High Z
Program	Active Low	High	High	High	Outputs
Program Verify	High	High	High	High	Outputs
Program Erase	High	High	High	High	High Z

BLOCK DIAGRAM



PROGRAMMING

The programming specifications are described in the Data Catalog PROM/ROM Programming Instructions Section.

Absolute Maximum Ratings*

Temperature Under Bias -10°C to +80°C

Storage Temperature -65°C to +125°C

All Input or Output Voltages with

Respect to Ground +6V to -0.3V

V_{DD} Supply Voltage with Respect

to Ground During Program +25.5V to -0.3V

*COMMENTS: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress-rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC and AC Operating Conditions During Read

	2716	2716-1	2716-2	2716-S	2716-B
Temperature Range	0°C - 70°C	0°C - 70°C	0°C - 70°C	0°C - 70°C	0°C - 70°C
V _{CC} Power Supply(1,2)	5V ± 5%	5V ± 10%	5V ± 5%	5V ± 5%	5V ± 5%
V _{DD} Power Supply(3)	V _{CC}	V _{CC}	V _{CC}	V _{CC}	V _{CC}

READ OPERATION

D.C. and Operating Characteristics

Symbol	Parameter	Limits			Unit	Conditions
		Min.	Typ. ⁽³⁾	Max.		
I _{IL}	Input Load Current			10	μA	V _{IL} = 5.25V
I _{LO}	Output Leakage Current			10	μA	V _{OUT} = 5.25V
I _{PPS} (2)	V _{DD} Current			5	mA	V _{DD} = 5.25V
I _{CC1} (2)	V _{CC} Current (Standby)		10	25	mA	CE = V _{DD} , OE = V _{IL}
I _{CC2} (2)	V _{CC} Current (Active)		57	100	mA	OE = CE = V _{IL}
V _{IL}	Input Low Voltage	-0.1		0.8	V	
V _{IH}	Input High Voltage	2.0		V _{CC} + 1	V	
V _{OL}	Output Low Voltage			0.45	V	I _{OL} = 2.1 mA
V _{OH}	Output High Voltage	2.4			V	I _{OH} = -400 μA

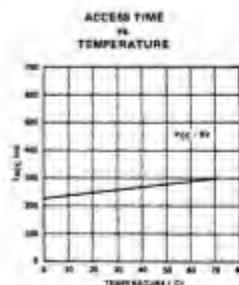
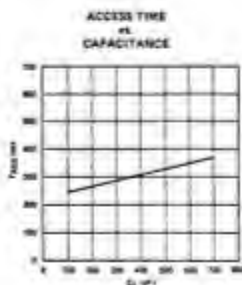
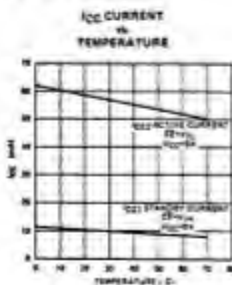
NOTES: 1. V_{CC} must be applied simultaneously or before V_{DD} and removed simultaneously or after V_{DD}.

2. V_{DD} may be connected directly to V_{CC} except during programming. The supply current would then be the sum of I_{CC} and I_{PPS}.

3. Typical values are for T_A = 25°C and nominal supply voltages.

4. This parameter is only tested and is not 100% tested.

Typical Characteristics



A.C. Characteristics

Symbol	Parameter	Limits (ns)										Test Condition
		2718		2716-1		2716-2		2716-3		2716-4		
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{ACC}	Address to Output Delay		450		350		290		450		450	$OE = OE = V_{IL}$
t_{CS}	CE to Output Delay		450		350		390		450		850	$OE = V_{IL}$
t_{OE}	Output Enable to Output Delay		120		120		120		100		200	$OE = V_{IL}$
t_{OH}	Output Enable High to Output From	0	100	0	100	0	100	0	100	0	100	$OE = V_{IL}$
t_{OL}	Output Hold from Address, CE or OE Whichever Occurs First	0		0		0		0		0		$OE = OE = V_{IL}$

Capacitance ⁽⁴⁾ $T_A = 25^\circ\text{C}$, $f = 1\text{ MHz}$

Symbol	Parameter	Typ.	Max.	Unit	Conditions
C_{IN}	Input Capacitance	4	8	pF	$V_{IN} = 0\text{V}$
C_{OUT}	Output Capacitance	8	12	pF	$V_{OUT} = 0\text{V}$

A.C. Test Conditions:

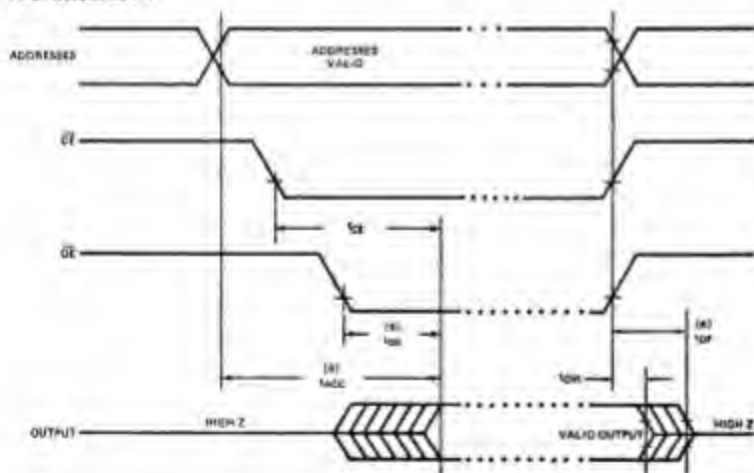
Output Load: 1 TTL gate and $C_L = 100\text{ pF}$ Input Rise and Fall Times: $<20\text{ ns}$

Input Pulse Levels: 0.8V to 2.2V

Timing Measurement Reference Level:

Inputs: 1V and 2V

Outputs: 0.8V and 2V

A.C. Waveforms ⁽¹⁾

NOTE

- V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously to after V_{PP} .
- V_{PP} may be connected directly to V_{CC} except during programming. The supply current would then be the sum of I_{CC} and I_{PP} .
- Typical values are for $T_A = 25^\circ\text{C}$ and nominal supply voltages.
- This parameter is only sampled and is not 100% tested.
- OE may be delayed up to $t_{AOC} - t_{OE}$ after the falling edge of CE without impact on t_{AOC} .
- t_{OE} is specified from OE or CE, whichever occurs first.

ERASURE CHARACTERISTICS

The erasure characteristics of the 2716 are such that erasure begins to occur when exposed to light with wavelength shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 2716 in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 2716 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 2716 window to prevent unintentional erasure.

The recommended erasure procedure lists Data Catalog PROM/ROM Programming Instruction Section for the 2716 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity \times exposure time) for erasure should be a minimum of 15 Wav/cm². The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μ W/cm² power rating. The 2716 should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

DEVICE OPERATION

The five modes of operation of the 2716 are listed in Table 1. It should be noted that all inputs for the five modes are at TTL levels. The power supplies required are a +5V V_{CC} and a V_{pp}. The V_{pp} power supply must be at 25V during the three programming modes, and must be at 5V in the other two modes.

TABLE 1. MODE SELECTION

MODE	CE Enable (pin 18)	OE (pin 20)	V _{pp} (25V)	V _{CC} (5V)	Outputs (16 bits)
Read	\overline{CE}	\overline{OE}	—	—	16 bits
Standby	\overline{CE}	\overline{OE}	—	—	High Z
Program	Active Low	Active Low	25V	5V	16 bits
Program Verify	\overline{CE}	\overline{OE}	25V	5V	16 bits
Program Erase	\overline{CE}	—	25V	5V	16 bits

READ MODE

The 2716 has two control functions, both of which must be logically satisfied in order to obtain data at the outputs. Chip Enable (CE) is the power control and should be used for device selection. Output Enable (OE) is the output control and should be used to gate data to the output pins, independent of device selection. Assuming that addresses are stable, address access time (t_{ACC}) is equal to the delay from CE to output (t_{CE}). Data is available at the outputs 120 ns (t_{OE}) after the falling edge of OE, assuming that CE has been low and addresses have been stable for at least t_{ACC} — t_{OE} .

STANDBY MODE

The 2716 has a standby mode which reduces the active power dissipation by 75%, from 505 mW to 132 mW. The 2716 is placed in the standby mode by applying a TTL high signal to the CE input. When in standby mode, the outputs are in a high impedance state, independent of the OE input.

OUTPUT OR-TIEING

Because 2716s are usually used in larger memory arrays, Intel has provided a 3 line control function that accommodates this use of multiple memory connections. The two line control function allows for:

- all the lowest possible memory power dissipation, and
- of complete assurance that output bus contention will not occur.

To most efficiently use these two control lines, it is recommended that \overline{CE} (pin 18) be decoded and used as the primary device selecting function, while \overline{OE} (pin 20) be made a common connection to all devices in the array and connected to the READ line from the system control bus. This assures that all deselected memory devices are in their low power standby mode and that the output pins are only active when data is desired from a particular memory device.

PROGRAMMING

Initially, and after each erasure, all bits of the 2716 are in the "1" state. Data is introduced by selectively programming "0"s into the desired bit locations. Although only "0"s will be programmed, both "1"s and "0"s can be presented in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The 2716 is in the programming mode when the V_{pp} power supply is at 25V and OE is at V_{pp}. The data to be programmed is applied 8 bits in parallel to the data output pins. The levels required for the address and data inputs are TTL.

When the address and data are stable, a 60 msec, active high, TTL program pulse is applied to the CE/PGM input. A program pulse must be applied at each address location to be programmed. You can program any location at any time — either individually, sequentially, or at random. The program pulse has a maximum width of 65 msec. The 2716 must not be programmed with a OC signal applied to the CE/PGM input.

Programming of multiple 2716s in parallel with the same data can be easily accomplished due to the simplicity of the programming requirements. Like inputs of the parallel 2716s may be connected together when they are programmed with the same data. A high level TTL pulse applied to the CE/PGM input programs the parallelized 2716s.

PROGRAM VERIFY

Programming of multiple 2716s in parallel with different data is also easily accomplished. Except for CE/PGM, all like inputs (including OE) of the parallel 2716s may be common. A TTL level program pulse applied to a 2716's CE/PGM input with V_{pp} at 25V will program that 2716. A low level CE/PGM input inhibits the other 2716 from being programmed.

PROGRAM VERIFY

A verify should be performed on the programmed bits to determine that they were correctly programmed. The verify may be performed with V_{pp} at 25V. Except during programming and program verify, V_{pp} must be at 5V.

Appendix C3



2102A, 2102AL/8102A-4* 1K x 1 BIT STATIC RAM

P/N	Standby Pwr. (mW)	Operating Pwr. (mW)	Access (ns)
2102AL-4	35	174	450
2102AL	35	174	350
2102AL-2	42	342	250
2102A-2	—	342	250
2102A	—	289	350
2102A-4	—	289	450

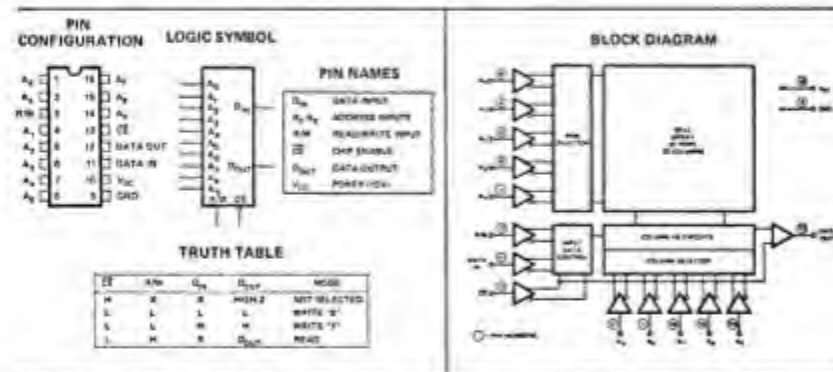
- Single +5 Volts Supply Voltage
- Directly TTL Compatible: All Inputs and Output
- Standby Power Mode (2102AL)
- Three-State Output: OR-Tie Capability
- Inputs Protected: All Inputs Have Protection Against Static Charge
- Low Cost Packaging: 16 Pin Dual-In-Line Configuration

The Intel® 2102A is a high speed 1024 word by one bit static random access memory element using N-channel MOS devices integrated on a monolithic array. It uses fully DC stable (static) circuitry and therefore requires no clocks or refreshing to operate. The data is read out nondestructively and has the same polarity as the input data.

The 2102A is designed for memory applications where high performance, low cost, large bit storage, and simple interfacing are important design objectives. A low standby power version (2102AL) is also available. It has all the same operating characteristics of the 2102A with the added feature of 35mW maximum power dissipation in standby and 174mW in operations.

It is directly TTL compatible in all respects: inputs, output, and a single +5 volt supply. A separate chip enable (CE) lead allows easy selection of an individual package when outputs are OR-tied.

The Intel® 2102A is fabricated with N-channel silicon gate technology. This technology allows the design and production of high performance easy to use MOS circuits and provides a higher functional density on a monolithic chip than either conventional MOS technology or P-channel silicon gate technology.



*All 8102A-4 specifications are identical to the 2102A-4 specifications.

Reprinted by permission of Intel Corporation Copyright © 1978

Absolute Maximum Ratings*

Ambient Temperature Under Bias	-10°C to 80°C
Storage Temperature	-65°C to +150°C
Voltage On Any Pin With Respect To Ground	-0.5V to +7V
Power Dissipation	1 Watt

*COMMENT:

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D. C. and Operating Characteristics

$T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$ unless otherwise specified.

Symbol	Parameter	2102A, 2102A-4 2102AL, 2102AL-4 Limits			2102A-2, 2102AL-2 Limits			Unit	Test Conditions
		Min.	Typ. (1)	Max.	Min.	Typ. (1)	Max.		
I_{LH}	Input Load Current		1	10		1	10	μA	$V_{DD} = 0$ to 5.25V
I_{LOH}	Output Leakage Current		1	5		1	5	μA	$\text{CE} = 2.0\text{V}$, $V_{OLH} = V_{OH}$
I_{LOL}	Output Leakage Current		-1	-10		-1	-10	μA	$\text{CE} = 2.0\text{V}$, $V_{OUT} = 0.4\text{V}$
I_{CC}	Power Supply Current		33	Note 2		49	65	mA	All inputs = 5.25V , Data Out Open, $T_A = 0^\circ\text{C}$
V_{IL}	Input Low Voltage	-0.5		0.8	-0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4			0.4	V	$I_{OL} = 2.1\text{mA}$
V_{OH}	Output High Voltage		2.4			2.4		V	$I_{OH} = -100\mu\text{A}$

Notes: 1. Typical values are for $T_A = 25^\circ\text{C}$ and nominal supply voltage.

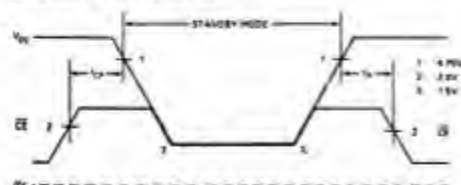
2. The maximum I_{CC} value is 55mA for the 2102A and 2102A-4, and 55mA for the 2102AL and 2102AL-4.

Standby Characteristics 2102AL, 2102AL-2, and 2102AL-4 (Available only in the Plastic Package)

$T_A = 0^\circ\text{C}$ to 70°C

Symbol	Parameter	2102AL, 2102AL-4 Limits			2102AL-2 Limits			Unit	Test Conditions
		Min.	Typ. (1)	Max.	Min.	Typ. (1)	Max.		
V_{DD}	V_{CC} in Standby	1.5			1.5			V	
$V_{CE}(2)$	CE Bias in Standby	2.0			2.0			V	$2.0\text{V} < V_{DD} < V_{CC} \text{ Max.}$
				V_{DD}			V_{DD}	V	$1.5\text{V} < V_{DD} < 2.0\text{V}$
I_{DD1}	Standby Current		15	23		20	28	mA	All inputs = $V_{DD1} = 1.5\text{V}$
I_{DD2}	Standby Current		20	30		25	36	mA	All inputs = $V_{DD2} = 2.0\text{V}$
t_{CP}	Chip Deassert to Standby Time	0			0			ns	
$t_{R}(2)$	Standby Recovery Time		t_{RC}			t_{RC}		ns	

STANDBY WAVEFORMS



NOTES:

- Typical values are for $T_A = 25^\circ\text{C}$.
- Consider the test conditions as shown: If the standby voltage (V_{DD}) is between 5.25V ($V_{CC} \text{ Max.}$) and 2.0V , then CE must be held at 2.0V Min. (V_{DD}). If the standby voltage is less than 2.0V but greater than 1.5V ($V_{DD} \text{ Min.}$), then CE and standby voltage must be at least the same value or, if they are different, CE must be the more positive of the two.
- $t_R = t_{RC}$ (READ CYCLE TIME)

2102A FAMILY

A. C. Characteristics $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$ unless otherwise specified

READ CYCLE

Symbol	Parameter	2102A-2, 2102AL-2		2102A, 2102AL		2102A-4, 2102AL-4	
		Min.	Max.	Min.	Max.	Min.	Max.
t_{RC}	Read Cycle	250		350		450	
t_{A}	Access Time		250		350		450
t_{CO}	Chip Enable to Output Time		130		180		230
t_{OH1}	Previous Read Data Valid with Respect to Address	40		40		40	
t_{OH2}	Previous Read Data Valid with Respect to Chip Enable	0		0		0	

WRITE CYCLE

t_{WC}	Write Cycle	250	350	450
t_{AW}	Address to Write Setup Time	20	20	20
t_{WP}	Write Pulse Width	180	250	300
t_{WR}	Write Recovery Time	0	0	0
t_{DW}	Data Setup Time	180	250	300
t_{DH}	Data Hold Time	0	0	0
t_{OW}	Chip Enable to Write Setup Time	180	250	300

A. C. CONDITIONS OF TEST

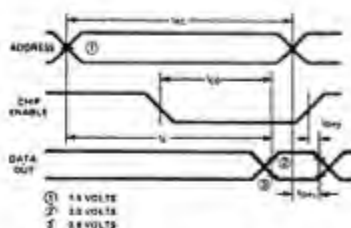
Input Pulse Levels: 0.8 Volt to 2.0 Volt
 Input Rise and Fall Times: 10ns
 Timing Measurements: Inputs: 1.5 Volts
 Reference Levels: Output: 0.8 and 2.0 Volts
 Output Load: 1 TTL Gate and $C_L = 100\text{ pF}$

Capacitance⁽²⁾ $T_A = 25^\circ\text{C}$, $f = 1\text{ MHz}$

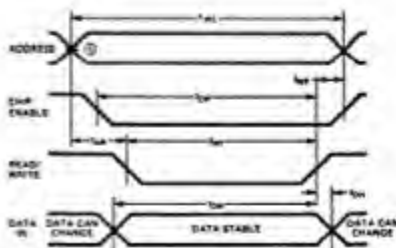
SYMBOL	TEST	LIMITS (pF)	
		TYP ⁽¹⁾	MAX.
C_{IN}	INPUT CAPACITANCE (ALL INPUT PINS) $V_{IN} = 0\text{V}$	3	5
C_{OUT}	OUTPUT CAPACITANCE $V_{OUT} = 0\text{V}$	7	10

Waveforms

READ CYCLE

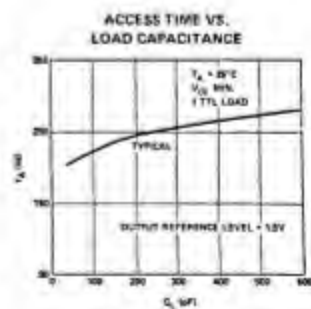
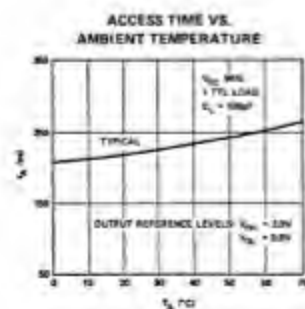
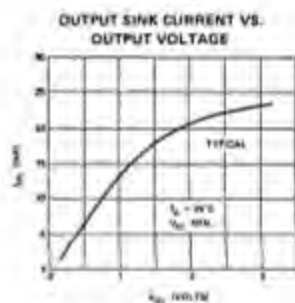
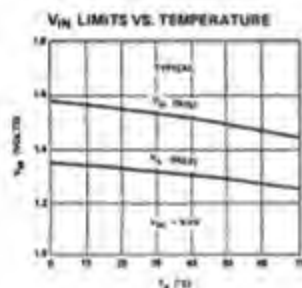
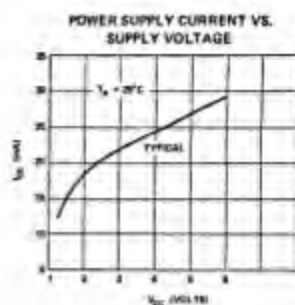
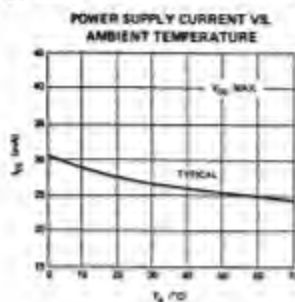


WRITE CYCLE



NOTES: 1. Typical values are for $T_A = 25^\circ\text{C}$ and nominal supply voltage.
 2. This parameter is periodically sampled and is not 100% tested.

Typical D. C. and A. C. Characteristics



Appendix C4



2114A 1024 X 4 BIT STATIC RAM

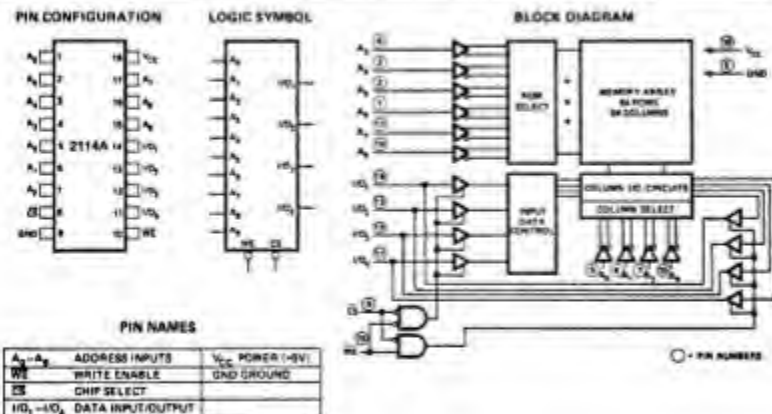
	2114AL-1	2114AL-2	2114AL-3	2114AL-4	2114A-4	2114A-5
Max. Access Time (ns)	100	120	150	200	200	250
Max. Current (mA)	40	40	40	40	70	70

- HMOS Technology
- Low Power, High Speed
- Identical Cycle and Access Times
- Single +5V Supply $\pm 10\%$
- High Density 18 Pin Package
- Completely Static Memory - No Clock or Timing Strobe Required
- Directly TTL Compatible: All Inputs and Outputs
- Common Data Input and Output Using Three-State Outputs
- 2114 Upgrade

The Intel® 2114A is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using HMOS, a high performance MOS technology. It uses fully DC stable (static) circuitry throughout, in both the array and the decoding, therefore it requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

The 2114A is designed for memory applications where the high performance and high reliability of HMOS, low cost, large bit storage, and simple interfacing are important design objectives. The 2114A is placed in an 18-pin package for the highest possible density.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select (CS) lead allows easy selection of an individual package when outputs are on-line.



ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	-10°C to 80°C
Storage Temperature	-65°C to 150°C
Voltage on any Pin	
With Respect to Ground	-3.5V to +7V
Power Dissipation	1.0W
D.C. Output Current	5mA

*COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND OPERATING CHARACTERISTICS

$T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 10\%$, unless otherwise noted.

SYMBOL	PARAMETER	2114AL-1/L-2/L-3/L-4		2114A-4/-5		UNIT	CONDITIONS
		Min.	Typ.(1)	Max.	Min.		
I_{LI}	Input Load Current (All Input Pins)		10		10	μA	$V_{IH} = 0$ to 5.5V
I_{LOI}	I/O Leakage Current		10		10	μA	$\overline{CS} = V_{IH}$ $V_{I/O} = \text{GND to VCC}$
I_{CC}	Power Supply Current	25	40	50	70	mA	$V_{OH} = \text{max.}$, $I_{I/O} = 0$ mA, $T_A = 0^\circ\text{C}$
V_{IL}	Input Low Voltage	-3.0	0.8	-3.0	0.8	V	
V_{IH}	Input High Voltage	2.0	6.0	2.0	6.0	V	
I_{OL}	Output Low Current	2.1	9.0	2.1	9.0	mA	$V_{OL} = 0.4\text{V}$
I_{OH}	Output High Current	-1.0	-2.5	-1.0	-2.5	mA	$V_{OH} = 2.4\text{V}$
$I_{OS}(2)$	Output Short Circuit Current		40		40	mA	

NOTE: 1. Typical values are for $T_A = 25^\circ\text{C}$ and $V_{CC} = 5.0\text{V}$.

2. Duration not to exceed 30 seconds.

CAPACITANCE

$T_A = 25^\circ\text{C}$, $f = 1.0\text{ MHz}$

SYMBOL	TEST	MAX	UNIT	CONDITIONS
C_{IO}	Input/Output Capacitance	5	pF	$V_{I/O} = 0\text{V}$
C_{IN}	Input Capacitance	5	pF	$V_{IH} = 0\text{V}$

NOTE: This parameter is periodically sampled and not 100% tested.

A.C. CONDITIONS OF TEST

Input Pulse Levels	0.5 Volt to 2.0 Volt
Input Rise and Fall Times	10 nsec
Input and Output Timing Levels	1.5 Volts
Output Load	1 TTL Gate and $C_L = 100\text{ pF}$

A.C. CHARACTERISTICS $T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5V \pm 10\%$, unless otherwise noted.READ CYCLE ^[1]

SYMBOL	PARAMETER	2114AL-1		2114AL-2		2114AL-3		2114A-4/L-4		2114A-5		UNIT
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{RC}	Read Cycle Time	100		120		150		200		250		ns
t_A	Access Time		100		120		150		200		250	ns
t_{CS}	Chip Selection to Output Valid		70		70		70		70		85	ns
t_{CO}	Chip Selection to Output Active		10		10		10		10		10	ns
t_{OH}	Output 3-state from Deselection		30		35		40		50		60	ns
t_{HOL}	Output Hold from Address Change		15		15		15		15		15	ns

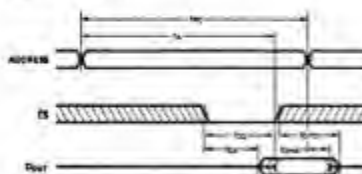
WRITE CYCLE ^[2]

SYMBOL	PARAMETER	2114AL-1		2114AL-2		2114AL-3		2114A-4/L-4		2114A-5		UNIT
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{WC}	Write Cycle Time		100		120		150		200		250	ns
t_W	Write Time		75		75		90		120		135	ns
t_{WR}	Write Release Time		0		0		0		0		0	ns
t_{OW}	Output 3-state from Write		30		35		40		50		60	ns
t_{WD}	Data to Write Time Overlap		70		70		90		120		135	ns
t_{WH}	Data Hold from Write Time		0		0		0		0		0	ns

NOTES:

1. A Read occurs during the overlap of a low \overline{CS} and a high \overline{WE} .
2. A Write occurs during the overlap of a low \overline{CS} and a low \overline{WE} . t_W is measured from the onset of \overline{CS} or \overline{WE} going low to the onset of \overline{CS} or \overline{WE} going high.

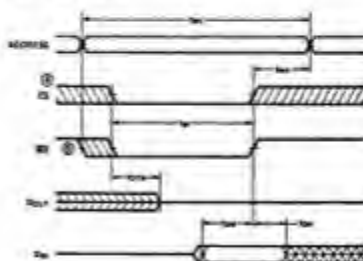
WAVEFORMS

READ CYCLE ^[3]

NOTES:

3. \overline{WE} is high for a Read Cycle.
4. If the \overline{CS} low transition occurs simultaneously with the \overline{WE} low transition, the output buffers remain in a high impedance state.
5. \overline{WE} must be high during all address transitions.

WRITE CYCLE



Appendix C5



8212 8-BIT INPUT/OUTPUT PORT

- Fully Parallel 8-Bit Data Register and Buffer
- Service Request Flip-Flop for Interrupt Generation
- Low Input Load Current — .25mA Max.
- Three State Outputs
- Outputs Sink 15mA
- 3.65V Output High Voltage for Direct Interface to 8008, 8080A, or 8085A CPU
- Asynchronous Register Clear
- Replaces Buffers, Latches and Multiplexers in Microcomputer Systems
- Reduces System Package Count

The 8212 input/output port consists of an 8-bit latch with 3-state output buffers along with control and device selection logic. Also included is a service request flip-flop for the generation and control of interrupts to the microprocessor. The device is multimode in nature. It can be used to implement latches, gated buffers or multiplexers. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with this device.

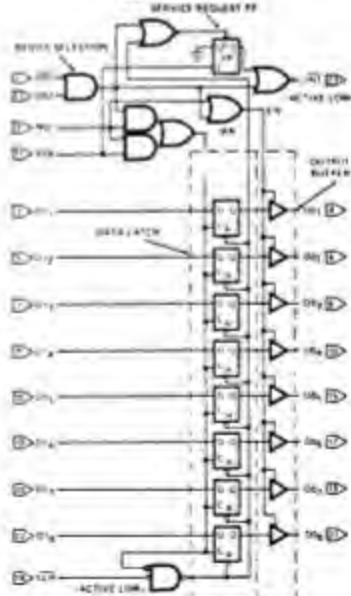
PIN CONFIGURATION



PIN NAMES

IN, DN	DATA IN
DN, DN	DATA OUT
DS, DS	SERVICE SELECT
MS	MODE
STR	STROBE
INT	INTERUPT (ACTIVE LOW)
CLR	CLEAR (ACTIVE LOW)

LOGIC DIAGRAM



FUNCTIONAL DESCRIPTION

Data Latch

The 8 flip-flops that make up the data latch are of a "D" type design. The output (Q) of the flip-flop will follow the data input (D) while the clock input (C) is high. Latching will occur when the clock (C) returns low.

The latched data is cleared by an asynchronous reset input (CLR). (Note: Clock (C) Overrides Reset (CLR).)

Output Buffer

The outputs of the data latch (Q) are connected to 3-state, non-inverting output buffers. These buffers have a common control line (EN); this control line either enables the buffer to transmit the data from the outputs of the data latch (Q) or disables the buffer, forcing the output into a high impedance state. (3-state).

The high-impedance state allows the designer to connect the 8212 directly onto the microprocessor bi-directional data bus.

Control Logic

The 8212 has control inputs $\overline{DS1}$, $\overline{DS2}$, MD and STB. These inputs are used to control device selection, data latching, output buffer state and service request flip-flop.

 $\overline{DS1}$, $\overline{DS2}$ (Device Select)

These 2 inputs are used for device selection. When $\overline{DS1}$ is low and $\overline{DS2}$ is high ($\overline{DS1} - \overline{DS2}$), the device is selected. In the selected state the output buffer is enabled and the service request flip-flop (SR) is asynchronously set.

MD (Mode)

This input is used to control the state of the output buffer and to determine the source of the clock input (C) to the data latch.

When MD is high (output mode) the output buffers are enabled and the source of clock (C) to the data latch is from the device selection logic ($\overline{DS1} - \overline{DS2}$).

When MD is low (input mode) the output buffer state is determined by the device selection logic ($\overline{DS1} - \overline{DS2}$) and the source of clock (C) to the data latch is the STB (Strobe) input.

STB (Strobe)

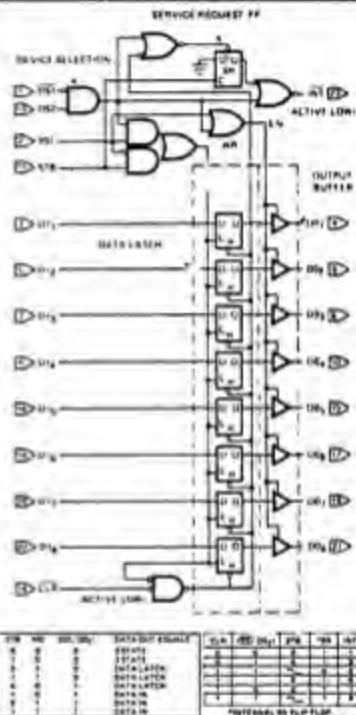
This input is used as the clock (C) to the data latch for the input mode MD = 0; and to asynchronously reset the service request flip-flop (SR).

Note that the SR flip-flop is negative edge triggered.

Service Request Flip-Flop

The (SR) flip-flop is used to generate and control interrupts in microcomputer systems. It is asynchronously set by the CLR input (active low). When the (SR) flip-flop is set it is in the non-interrupting state.

The output of the (SR) flip-flop (Q) is connected to an inverting input of a "NOR" gate. The other input to the "NOR" gate is non-inverting and is connected to the device selection logic ($\overline{DS1} - \overline{DS2}$). The output of the "NOR" gate (INT) is active low (interrupting state) for connection to active low input priority generating circuits.



Applications of the 8212 — For Microcomputer Systems

- I Basic Schematic Symbol
- II Gated Buffer
- III Bi-Directional Bus Driver
- IV Interrupting Input Port

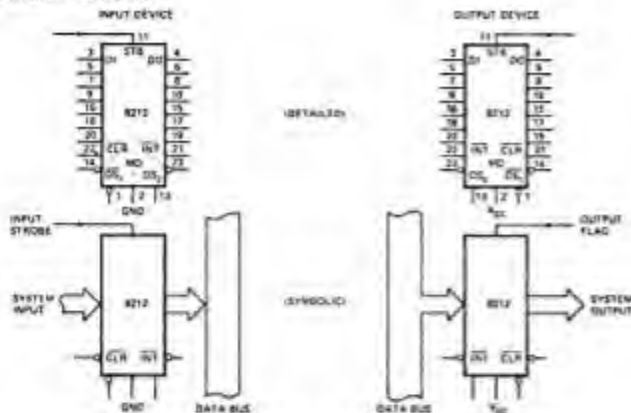
- V Interrupt Instruction Port
- VI Output Port
- VII 8085A Status Latch
- VIII 8085A Address Latch

1. Basic Schematic Symbols

Two examples of ways to draw the 8212 on system schematics — (1) the top being the detailed view showing pin numbers, and (2) the bottom being the symbolic view

showing the system input or output as a system bus (bus containing 8 parallel lines). The output to the data bus is symbolic in referencing 8 parallel lines.

BASIC SCHEMATIC SYMBOLS



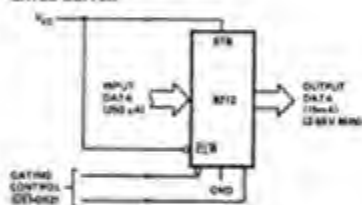
II. Gated Buffer (3-State)

The simplest use of the 8212 is that of a gated buffer. By tying the mode signal low and the strobe input high, the data latch is acting as a straight through gate. The output buffers are then enabled from the device selection logic DS1 and DS2.

When the device selection logic is false, the outputs are 3-state.

When the device selection logic is true, the input data from the system is directly transferred to the output. The input data load is 250 micro amps. The output data can sink 13 milli amps. The minimum high output is 3.65 volts.

GATED BUFFER

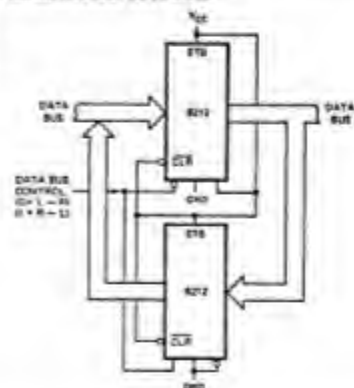


8212

III. Bi-Directional Bus Driver

A pair of 8212's wired (back-to-back) can be used as a symmetrical drive, bi-directional bus driver. The devices are controlled by the data bus input control which is connected to DS1 on the first 8212 and DS2 on the second. One device is active, and acting as a straight through buffer the other is in 3-state mode. This is a very useful circuit in small system design.

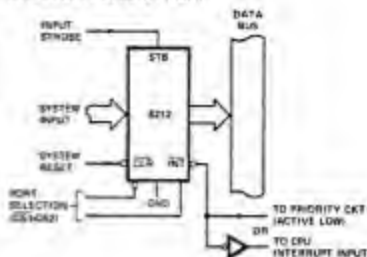
BI-DIRECTIONAL BUS DRIVER



IV. Interrupting Input Port

This use of an 8212 is that of a system input port that accepts a strobe from the system input source, which in turn clears the service request flip-flop and interrupts the processor. The processor then goes through a service routine, identifies the port, and causes the device selection logic to go true — enabling the system input data onto the data bus.

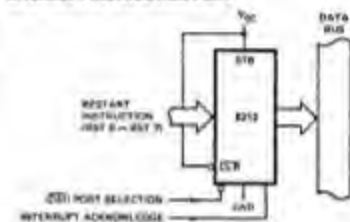
INTERRUPTING INPUT PORT



V. Interrupt Instruction Port

The 8212 can be used to gate the interrupt instruction, normally RESTART instructions, onto the data bus. The device is enabled from the interrupt acknowledge signal from the microprocessor and from a port selection signal. This signal is normally tied to ground. (DS1 could be used to multiplex a variety of interrupt instruction ports onto a common bus.)

INTERRUPT INSTRUCTION PORT

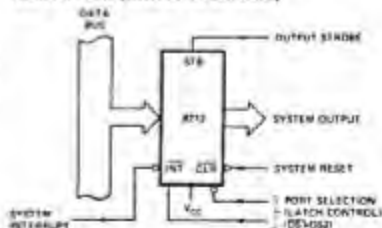


8212

VI. Output Port (With Hand-Shaking)

The 8212 can be used to transmit data from the data bus to a system output. The output strobe could be a hand-shaking signal such as "reception of data" from the device that the system is outputting to. In turn, an interrupt from the system signifying the reception of data. The selection of the port comes from the device selection logic. (DS1 - DS2.)

OUTPUT PORT (WITH HAND-SHAKING)

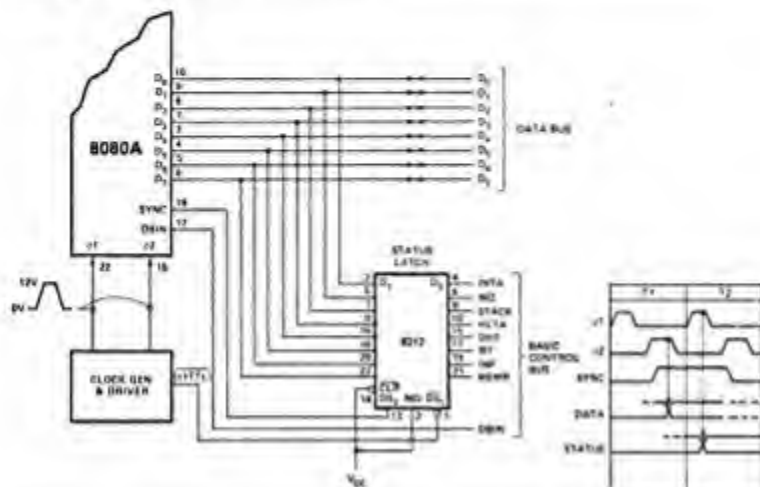


VII. 8080A Status Latch

Here the 8212 is used as the status latch for an 8080A microcomputer system. The input to the 8212 latch is directly from the 8080A data bus. Timing shows that when the SYNC signal is true, which is connected to the CS2 input and the phase 1 signal is true, which is a TTL level coming from the clock generator, then the status data will be latched into the 8212.

Note: The mode signal is tied high so that the output on the latch is active and enabled all the time.

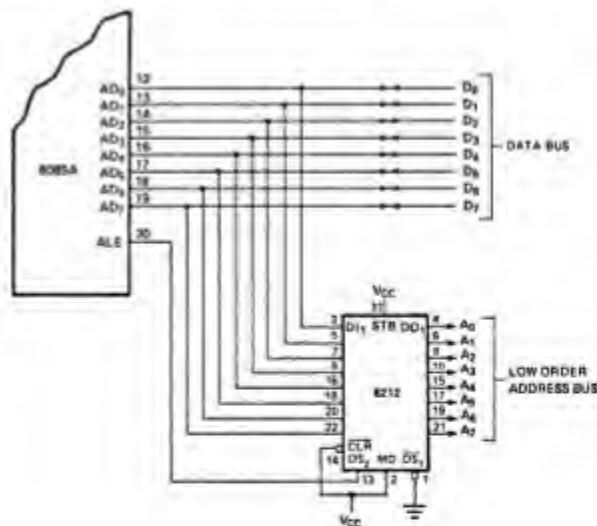
It is shown that the two areas of concern are the bi-directional data bus of the microprocessor and the control bus.



8212

VIII. 8085A Low-Order Address Latch

The 8085A microprocessor uses a multiplexed address/data bus that contains the low order 8-bits of address information during the first part of a machine cycle. The same bus contains data at a later time in the cycle. An address latch enable (ALE) signal is provided by the 8085A to be used by the 8212 to latch the address so that it may be available through the whole machine cycle. Note in this configuration, the MODE input is tied high, keeping the 8212's output buffers turned on at all times.



ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias Plastic 0°C to $+70^{\circ}\text{C}$
 Storage Temperature -55°C to $+160^{\circ}\text{C}$
 All Output or Supply Voltages -0.5 to $+7$ Volts
 All Input Voltages $+3.0$ to 5.5 Volts
 Output Currents 100mA

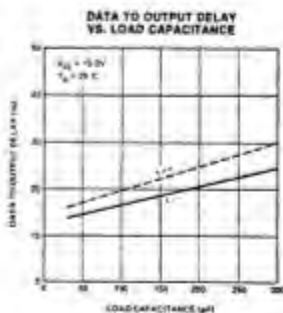
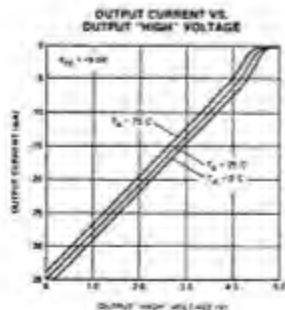
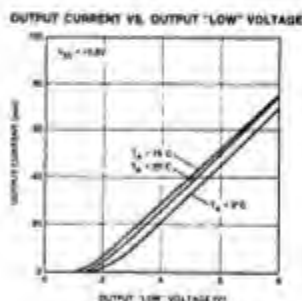
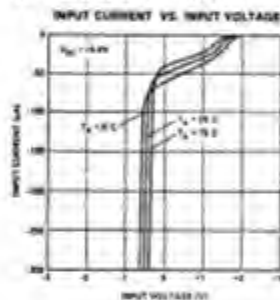
*COMMENT

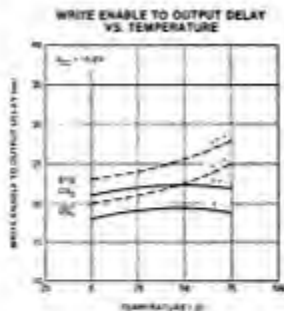
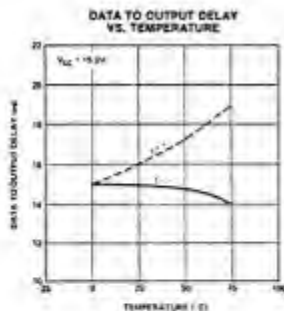
Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS $T_A = 0^{\circ}\text{C}$ to $+70^{\circ}\text{C}$, $V_{CC} = +5\text{V} \pm 5\%$

Symbol	Parameter	Limits		Unit	Test Conditions
		Min.	Typ.		
I_F	Input Load Current, ACK, DS, CR, DIN-Data Inputs			-25	$m\text{A}$, $V_F = .45\text{V}$
I_F	Input Load Current MD Input			-75	$m\text{A}$, $V_F = .45\text{V}$
I_F	Input Load Current DS+ Input			-1.0	$m\text{A}$, $V_F = .45\text{V}$
I_L	Input Leakage Current, ACK, DS, CR, DIN-Data Inputs			10	μA , $V_{IN} \leq V_{CC}$
I_L	Input Leakage Current MD Input			30	μA , $V_{IN} \leq V_{CC}$
I_L	Input Leakage Current DS+ Input			40	μA , $V_{IN} \leq V_{CC}$
V_C	Input Forward Voltage Clamp			-1	V , $I_C = -5\text{mA}$
V_{IL}	Input "Low" Voltage			.65	V
V_{IH}	Input "High" Voltage	2.0		V	
V_{OL}	Output "Low" Voltage			.45	V , $I_{OL} = 15\text{mA}$
V_{OH}	Output "High" Voltage	3.65	4.0	V	$I_{OH} = -13\text{mA}$
I_{SC}	Short Circuit Output Current	-15		-75	$m\text{A}$, $V_O = 0\text{V}$, $V_{CC} = 5\text{V}$
I_{OL}	Output Leakage Current High Impedance State			20	μA , $V_O = .45\text{V}/5.25\text{V}$
I_{CC}	Power Supply Current		90	130	$m\text{A}$

TYPICAL CHARACTERISTICS





8212

A.C. CHARACTERISTICS $T_A = 0^\circ\text{C}$ to $+70^\circ\text{C}$, $V_{CC} = +5\text{V} \pm 5\%$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
t_{PW}	Pulse Width	20			ns	
t_{DQ}	Data to Output Delay			30	ns	Note 1
t_{WE}	Write Enable to Output Delay			40	ns	Note 1
t_{DSR}	Data Set Up Time	15			ns	
t_H	Data Hold Time	20			ns	
t_R	Reset to Output Delay			40	ns	Note 1
t_S	Set to Output Delay			30	ns	Note 1
t_E	Output Enable/Disable Time			45	ns	Note 1
t_C	Clear to Output Delay			50	ns	Note 1

CAPACITANCE* $F = 1\text{MHz}$, $V_{BIAS} = 2.5\text{V}$, $V_{CC} = +5\text{V}$, $T_A = 25^\circ\text{C}$

Symbol	Test	Limits	
		Typ.	Max.
C_{IN}	DS1 MD Input Capacitance	9pF	12pF
C_{IN}	DS2 CK, ACK, Cln-Dig Input Capacitance	5pF	8pF
C_{OUT}	DO1-DO3 Output Capacitance	8pF	12pF

*This parameter is sampled and not 100% tested.

SWITCHING CHARACTERISTICS

Conditions of Test

Input Pulse Amplitude = 2.5V

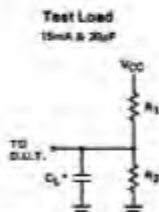
Input Rise and Fall Times 5ns

Between 1V and 2V Measurements made at 1.5V with 15mA and 30pF Test Load

Note 1:

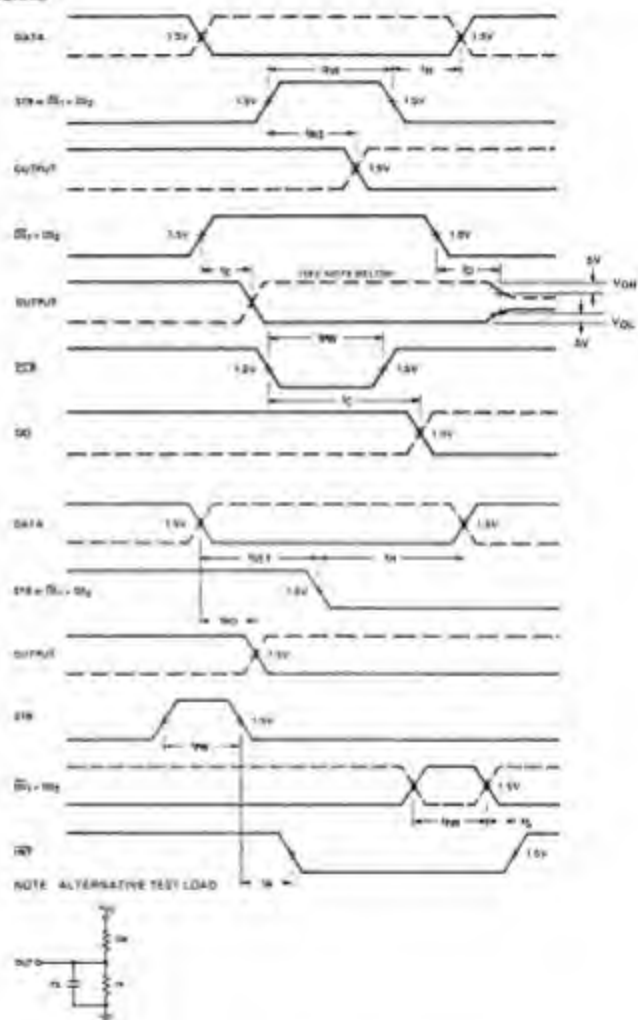
Test	C_L^*	R_1	R_2
t_{DQ} , t_{WE} , t_R , t_S , t_C	30pF	300Ω	600Ω
t_E , ENABLE↑	50pF	10KΩ	1KΩ
t_E , ENABLE↓	30pF	300Ω	600Ω
t_E , DISABLE↑	5pF	300Ω	600Ω
t_E , DISABLE↓	5pF	10KΩ	1KΩ

*Includes probe and jig capacitance



*INCLUDING JIG & PROBE CAPACITANCE

TIMING DIAGRAM



MAXIMUM GUARANTEED RATINGS†

Operating Temperature Range	0° C to +70° C
Storage Temperature Range	-65° C to +150° C
GND and V _{DD} , with respect to V _{CC}	-20V to +0.3V
Logic Input Voltages, with respect to V _{CC}	-20V to +0.3V

† Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

ELECTRICAL CHARACTERISTICS

(T_A = 0° C to +70° C, V_{CC} = +5V ±0.5V, V_{DD} = -12V ±1.0V, unless otherwise noted)

Characteristic	Min	Typ	Max	Unit	Conditions
CLOCK	20	50	100	KHz	see fig. 1 footnote (**) for typical R-C values
DATA INPUT					
Logic "0" Level			+0.8	V	
Logic "1" Level	V _{CC} -1.5			V	
Input Capacitance			10	pf	
INPUT CURRENT					
*Control, Shift & Y ₀ thru Y ₁₀	10	100	140	μA	V _{IN} = +5.0V
*Control, Shift & Y ₀ thru Y ₁₀	5	30	50	μA	V _{IN} = Ground
Data Invert, Parity Invert		.01	1	μA	V _{IN} = -5.0V to +5.0V
DATA OUTPUT & X OUTPUT					
Logic "0" Level			+0.4	V	I _{OL} = 1.6mA (see fig. 7)
Logic "1" Level	V _{CC} -1.0			V	I _{OH} = 100 μA
POWER CONSUMPTION		140	200	mW	Nom. Power Supp. Voltages (see fig. 8)
SWITCH CHARACTERISTICS					
Minimum Switch Closure	see timing diagram-fig. 2				
Contact Closure Resistance between X1 and Y1			300	Ohm	
Contact Open Resistance between X1 and Y1	1 × 10 ⁶			Ohm	

*Inputs with internal Resistor to V_{DD}

DESCRIPTION OF OPERATION

The KR2376-XX contains (see Fig. 1), a 2376-bit ROM, 8-stage and 11-stage ring counters, an 11-bit comparator, an oscillator circuit, an externally controllable delay network for eliminating the effect of contact bounce, and TTL/DTL/MOS compatible output drivers.

The ROM portion of the chip is a 264 by 9-bit memory arranged into three 88-word by 9-bit groups. The appropriate levels on the Shift and Control inputs selects one of the three 88-word groups; the 88-individual word locations are addressed by the two ring counters. Thus, the ROM

address is formed by combining the Shift and Control inputs with the two ring counters.

The external outputs of the 8-stage ring counter and the external inputs to the 11-bit comparator are wired to the keyboard to form an X-Y matrix with the 88-keyboard switches as the crosspoints. In the standby condition, when no key is depressed, the two ring counters are clocked and sequentially address the ROM; the absence of a Strobe Output indicates that the Data Outputs are 'not valid' at this time.

When a key is depressed, a single path is completed between one output of the 8-stage ring counter (X0 thru X7) and one input of the 11-bit comparator (Y0-Y10). After a number of clock cycles, a condition will occur where a level on the selected path to the comparator matches a level on the corresponding comparator input from the 11-stage ring counter. When this occurs, the comparator generates a signal to the clock control and to the Strobe Output (via the delay network). The clock control stops the clocks to the ring counters and the Data Outputs

(B1-B9) stabilize with the selected 9-bit code, indicated by a 'valid' signal on the Strobe Output. The Data Outputs remain stable until the key is released.

As an added feature two inputs are provided for external polarity control of the Data Outputs. Parity Invert (pin 6) provides polarity control of the Parity Output (pin 7) while the Data and Strobe Invert Input (pin 20) provides for polarity control of Data Outputs B1 thru B6 (pins 3 thru 15) and the Strobe Output (pin 16).

SPECIAL PATTERNS

Since the selected coding of each key is defined during the manufacture of the chip, the coding can be changed to fit any particular application of the keyboard. Up to 264 codes of up to 8 bits (plus one parity bit) can be programmed into the KR2376-XX

ROM covering most popular codes such as ASCII, EBCDIC, Selectric, etc., as well as many specialized codes. The ASCII code is available as a standard pattern. For special patterns, use Fig. 9.

TIMING DIAGRAM

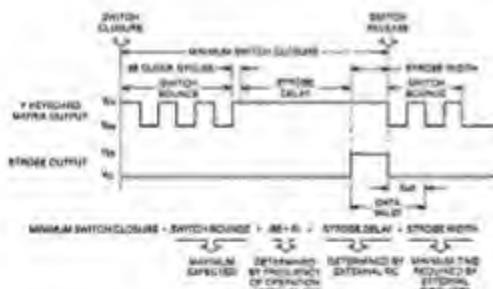
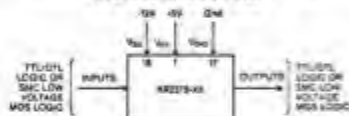


Fig. 2

POWER SUPPLY CONNECTIONS FOR TTL/DTL OPERATION



POWER SUPPLY CONNECTIONS FOR MOS OPERATION

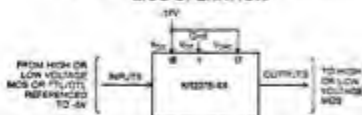
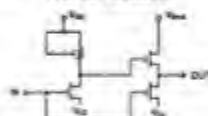


Fig. 3

OUTPUT DRIVER & "X" OUTPUT STAGE TO KEYBOARD



"Y" INPUT STAGE FROM KEYBOARD

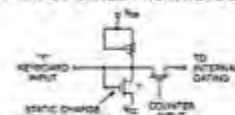


Fig. 4

Appendix C8

STANDARD MICROSYSTEMS CORPORATION

25 Marston Blvd., Norwalk, CT 06854
 (203) 270-2000 FAX: (203) 270-2000
 We keep ahead of our competition so you can keep ahead of yours.

**CRT 5027
 CRT 5037
 CRT 5057***
μPC FAMILY

CRT Video Timer and Controller VTAC®

FEATURES

- ☐ Fully Programmable Display Format
 - Characters per data row (1-200)
 - Data rows per frame (1-64)
 - Raster scans per data row (1-16)
- ☐ Programmable Monitor Sync Format
 - Raster Scans/Frame (250-1023)
 - "Front Porch"
 - Sync Width
 - "Back Porch"
 - Interlace/Non-Interlace
 - Vertical Blanking
- ☐ Lock Line Input (CRT 5057)
- ☐ Direct Outputs to CRT Monitor
 - Horizontal Sync
 - Vertical Sync
 - Composite Sync (CRT 5027, CRT 5037)
 - Blanking
 - Cursor coincidence
- ☐ Programmed via:
 - Processor data bus
 - External PROM
 - Mask Option ROM
- ☐ Standard or Non-Standard CRT Monitor Compatible
- ☐ Refresh Rate: 60Hz, 50Hz, ...
- ☐ Scrolling
 - Single Line
 - Multi-Line
- ☐ Cursor Position Registers
 - Character Format: 3x7, 7x9, ...
- ☐ Programmable Vertical Data Positioning
- ☐ Balanced Beam Current Interlace (CRT 5037)
- ☐ Graphics Compatible

PIN CONFIGURATION



- ☐ Split-Screen Applications
 - Horizontal
 - Vertical
- ☐ Interlace or Non-Interlace operation
- ☐ TTL Compatibility
- ☐ BUS Oriented
- ☐ High Speed Operation
- ☐ CMOS/PMOS* N-Channel Silicon Gate Technology
- ☐ Compatible with CRT 5003 VDAC™
- ☐ Compatible with CRT 7004

GENERAL DESCRIPTION

The CRT Video Timer and Controller (VTAC) is a user-programmable 40-pin CMOS/PMOS* n-channel MOS/LSI device containing the logic functions required to generate all the timing signals for the presentation and formatting of interlaced and non-interlaced video data on a standard or non-standard CRT monitor.

With the exception of the dot counter, which may be clocked at a video frequency above 25 MHz and therefore not recommended for MOS implementation, all frame formatting, such as horizontal, vertical, and composite sync, characters per data row, data rows per frame, and raster scans per data row and per frame are totally user programmable. The data row counter has been designed to facilitate scrolling.

Programming is effected by loading seven 8-bit control registers directly off an 8-bit bidirectional data bus. Four register address lines and a chip select line provide complete microprocessor compatibility for program controlled setup. The device can be "self loaded" via an external PROM tied on the data bus as described in the OPERATION section. Formatting can also be programmed by a single mask option.

In addition to the seven control registers two additional registers are provided to store the cursor character and data row addresses for generation of the cursor video signal. The contents of these two registers can also be read out onto the bus for update by the program.

Three versions of the VTAC* are available. The CRT 5027 provides non-interlaced operation with an even or odd number of scan lines per data row, or interlaced operation with an even number of scan lines per data row. The CRT 5037 may be programmed for an odd or even number of scan lines per data row in both interlaced and non-interlaced modes. Programming the CRT 5037 for an odd number of scan lines per data row eliminates character distortion caused by the uneven beam current normally associated with odd field-even field interlacing of alphanumeric displays.

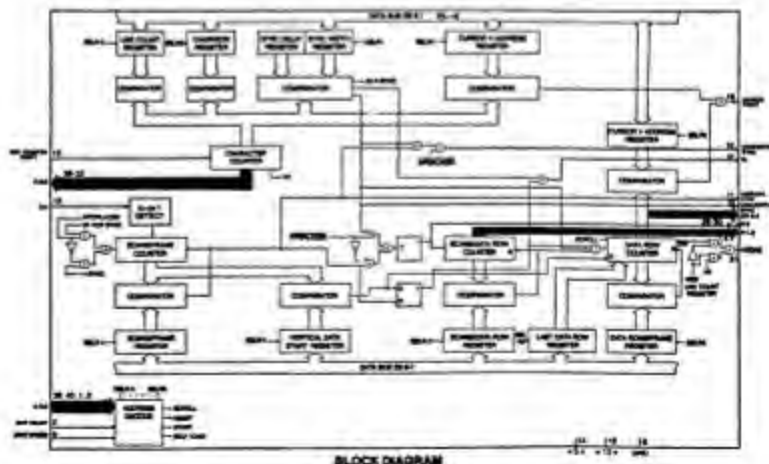
The CRT 5057 provides the ability to lock a CRT's vertical refresh rate, as controlled by the VTAC's vertical sync pulse, to the 50 Hz or 60 Hz line frequency thereby eliminating the so called "swim" phenomenon. This is particularly well suited for European system requirements. The line frequency waveform, processed to conform to the VTAC's specified logic levels, is applied to the line lock input. The VTAC* will inhibit generation of vertical sync until a zero to one transition on this input is detected. The vertical sync pulse is then initiated within one scan line after this transition rises above the logic threshold of the VTAC.*

To provide the pin required for the line lock input, the composite sync output is not provided in the CRT 5057.

*FOR FUTURE RELEASE

Description of Pin Functions

Pin No.	Symbol	Name	Input/ Output	Function
25-18	DBF-7	Data Bus	I/O	Data bus. Input bus for control words from microprocessor or PROM. Bidirectional bus for cursor address.
3	CS	Chip Select	I	Signals chip that it is being addressed
39, 40, 1, 2	AR-3	Register Address	I	Register address bits for selecting one of seven control registers or either of the cursor address registers
9	DS	Data Strobe	I	Strobes DBF-7 into the appropriate register or outputs the cursor character address or cursor line address onto the data bus
12	DCG	DOT Counter Carry	I	Carry from off chip dot counter establishing basic character clock rate. Character clock.
38-32	HCF-6	Character Counter Outputs	O	Character counter outputs.
7, 8, 4	R1-3	Scan Counter Counter Outputs	O	Three most significant bits of the Scan Counter; row select inputs to character generator.
31	H7/DR5	H7/DR5	O	Pin definition is user programmable. Output is MSB of Character Counter if horizontal line count (REG. #) is ≥ 128 ; otherwise output is MSB of Data Row Counter.
8	R6	Scan Counter LSB	O	Least significant bit of the scan counter. In the interlaced mode with an even number of scans per data row, R6 will toggle at the field rate; for an odd number of scans per data row in the interlaced mode, R6 will toggle at the data row rate.
26-30	DRF-4	Data Row Counter Outputs	O	Data Row counter outputs.
17	BL	Blank	O	Defines non active portion of horizontal and vertical scans.
15	HSYN	Horizontal Sync	O	Initiates horizontal retrace.
11	VSYN	Vertical Sync	O	Initiates vertical retrace.
10	CSYN/LLI	Composite Sync Output/ Line Lock Input	O/I	Composite sync is provided on the CRT 9027 and CRT 5037. This output is active in non-interlaced mode only. Provides a true RS-170 composite sync wave form. For the CRT 5057, this pin is the Line Lock input. The line frequency waveform, processed to conform to the VTAC SM specified logic levels, is applied to this pin.
16	CRV	Cursor Video	O	Defines cursor location in data field.
14	Vcc	Power Supply	PS	+5 volt Power Supply
13	Vcc	Power Supply	PS	+12 volt Power Supply



Operation

The design philosophy employed was to allow the device to interface effectively with either a microprocessor based or hardware logic system. The device is programmed by the user in one of two ways; via the processor data bus as part of the system initialization routine, or during power up via a PROM tied on the data bus and addressed directly by the Row Select outputs of the chip. (See figure 4). Seven 8 bit words are required to fully program the chip. Bit assignments for these words are shown in Table 1. The information contained in these seven words consists of the following:

Horizontal Formatting:	
Characters/Data Row	A 3 bit code providing 8 mask programmable character lengths from 20 to 132. The standard device will be masked for the following character lengths; 20, 32, 40, 64, 72, 80, 96, and 132.
Horizontal Sync Delay	3 bits assigned providing up to 8 character times for generation of "front porch"
Horizontal Sync Width	4 bits assigned providing up to 15 character times for generation of horizontal sync width
Horizontal Line Count	6 bits assigned providing up to 256 character times for total horizontal formatting.
Skew Bits	A 2 bit code providing from a 0 to 2 ¹ character skew (delay) between the horizontal address counter and the blank and sync (horizontal, vertical, composite) signals to allow for timing of video data prior to generation of composite video signal. The Cursor Video signal is also skewed as a function of this code.
Vertical Formatting:	
Interlaced/Non-interlaced	This bit provides for data presentation with address text formatting for interlaced systems. It modifies the vertical timing counters as described below. A logic 1 establishes the interlace mode.
Scans/Frame	8 bits assigned, defined according to the following equations: Let X = value of 8 assigned bits. 1) in interlaced mode—scans/frame = $2X + 513$. Therefore for 525 scans, program X = 0 (00000110). Vertical sync will occur precisely every 262.5 scans, thereby producing two interlaced fields. Range = 513 to 1023 scans/frame, odd counts only. 2) in non-interlaced mode—scans/frame = $2X + 256$. Therefore for 262 scans, program X = 3 (00000111). Range = 256 to 768 scans/frame, even counts only. In either mode, vertical sync width is fixed at three horizontal scans (= 3H).
Vertical Data Start	8 bits defining the number of raster scans from the leading edge of vertical sync until the start of display data. At this raster scan the data row counter is set to the data row address at the top of the page.
Data Rows/Frame	6 bits assigned providing up to 64 data rows per frame.
Last Data Row	6 bits to allow up or down scrolling via a preset defining the count of the last displayed data row.
Scans/Data Row	4 bits assigned providing up to 16 scan lines per data row.

Additional Features

Device Initialization:

Under microprocessor control—The device can be reset under system or program control by presenting a 1918 address on A3-8. The device will remain reset at the top of the even field page until a start command is executed by presenting a 1111 address on A3-8.

Via "Self Loading"—In a non-processor environment, the self loading sequence is effected by presenting and holding the 1111 address on A3-8, and is initiated by the receipt of the strobe pulse (DS). The 1111 address should be maintained long enough to insure that all seven registers have been loaded (in most applications under one millisecond). The timing sequence will begin one line scan after the 1111 address is removed. In processor based systems, self loading is initiated by presenting the 1111 address to the device. Self loading is terminated by presenting the start command to the device which also initiates the timing chain.

Scrolling—In addition to the Register 6 storage of the last displayed data row a "scroll" command (address 1911) presented to the device will increment the first displayed data row count to facilitate up scrolling in certain applications.

Register Selects/Command Codes

A3	A2	A1	A0	Select/Command	Description
0	0	0	0	Load Control Register 0	See Table 1
0	0	0	1	Load Control Register 1	
0	0	1	0	Load Control Register 2	
0	0	1	1	Load Control Register 3	
0	1	0	0	Load Control Register 4	
0	1	0	1	Load Control Register 5	
0	1	1	0	Load Control Register 6	
0	1	1	1	Processor Initiated Self Load	
1	0	0	0	Read Cursor Line Address	Resets timing chain to top left of page. Reset is latched on chip by DS and counters are held until released by scan command.
1	0	0	1	Read Cursor Character Address	
1	0	1	0	Reset	
1	0	1	1	Up Scroll	Increments address of first displayed data row on page, i.e. prior to receipt of scroll command—top line = 0, bottom line = 23. After receipt of Scroll Command—top line = 1, bottom line = 0.
1	1	0	0	Load Cursor Character Address*	Receipt of this command after a Reset or Processor Self Load command will release the timing chain approximately one scan line later. In applications requiring synchronous operation of more than one CRT 5027 the dot counter carry should be held low during the DS for this command.
1	1	0	1	Load Cursor Line Address*	
1	1	1	0	Start Timing Chain	
1	1	1	1	Non-Processor Self Load	Device will begin self load via PROM when DS goes low. The 1111 command should be maintained on A3-0 long enough to guarantee self load. (Scan counter should cycle through at least once). Self load is automatically terminated and timing chain initiated when the all "1's" condition is removed, independent of DS. For synchronous operation of more than one VTAC*, the Dot Counter Carry should be held low when the command is removed.

*NOTE: During Self-Load, the Cursor Character Address Register (REG 7) and the Cursor Row Address Register (REG 8) are enabled during states 0111 and 1000 of the RS-16 Scan Counter outputs respectively. Therefore, cursor data in the PROM should be stored at these addresses.

TABLE 1

BIT ASSIGNMENT CHART

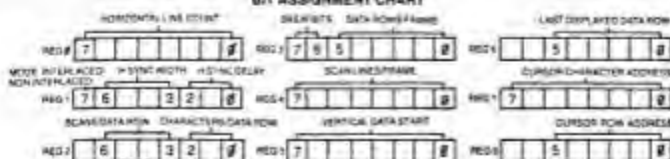


FIGURE 1 VIDEO TAPING

FIGURE 1 VIDEO TWINING

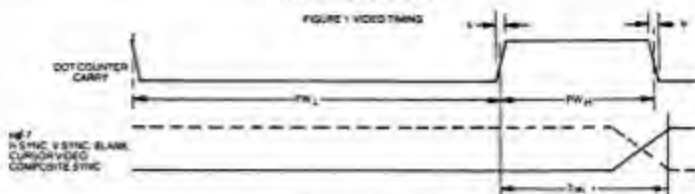


FIGURE 2. LOAD/READ TRENDS.

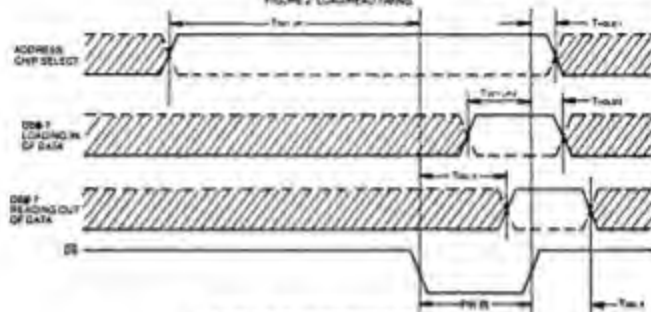
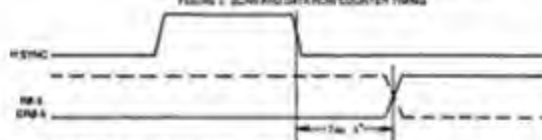
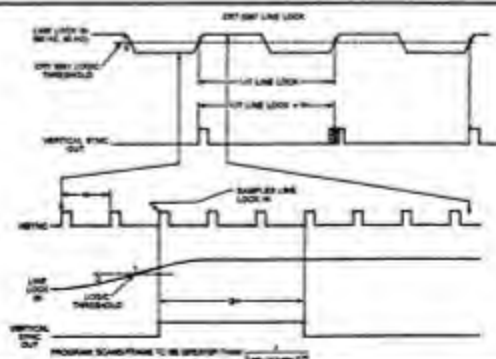


FIGURE 1. SCAN AND DATA ROW COUNTER TYPING



*AR-3 and DR-5 may change prior to the falling edge of H sync



MAXIMUM GUARANTEED RATINGS*

Operating Temperature Range	-55°C to +70°C
Storage Temperature Range	-55°C to +150°C
Lead Temperature (soldering, 10 sec)	+325°C
Positive Voltage on any Pin, with respect to ground	+18.0V
Negative Voltage on any Pin, with respect to ground	-0.3V

*Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

NOTE: When powering this device from laboratory or system power supplies, it is important that the Absolute Maximum Ratings not be exceeded or device failure can result. Some power supplies exhibit voltage spikes or "glitches" on their outputs when the AC power is switched on and off. In addition, voltage transients on the AC power line may appear on the DC output. For example, the bench power supply programmed to deliver +12 volts may have large voltage transients when the AC power is switched on and off. If this possibility exists it is suggested that a clamp circuit be used.

ELECTRICAL CHARACTERISTICS (T_a = 0°C to 70°C, V_{CC} = +5V ± 5%, V_{EE} = +12V ± 5%, unless otherwise noted)

Parameter	Min.	Typ.	Max.	Unit	Comments
D.C. CHARACTERISTICS					
INPUT VOLTAGE LEVELS					
Low Level, V _{IL}			0.8	V	
High Level, V _{IH}	V _{CC} - 1.5		V _{CC}	V	
OUTPUT VOLTAGE LEVELS					
Low Level—V _{OL} for RQ-3			0.4	V	I _{OL} = 3.2mA
Low Level—V _{OL} at others			0.4	V	I _{OL} = 1.6mA
High Level—V _{OH} for RQ-3, DRQ-7	2.4			V	I _{OH} = 80µA
High Level—V _{OH} at others	2.4			V	I _{OH} = 40µA
INPUT CURRENT					
Low Level, I _L (Address, CS only)			250	µA	V _{IN} = 0.4V
Leakage, I _L (All Inputs except Address, CS)			10	µA	0.5V _{CC} to V _{CC}
INPUT CAPACITANCE					
Data Bus, C _{IN}		10	15	pF	
CS, Clock, C _{IN}		25	40	pF	
All other, C _{IN}		10	15	pF	
DATA BUS LEAKAGE IN INPUT MODE					
I _{OL}			10	µA	0.4V ≤ V _{IN} ≤ 5.25V
POWER SUPPLY CURRENT					
I _{CC}		80	100	mA	
I _{EE}		40	70	mA	
A.C. CHARACTERISTICS					
DOT COUNTER CARRY					
Frequency	0.2		4.0	MHz	Figure 1
PW _H	35			ns	Figure 1
PW _L	215			ns	Figure 1
t _r , t _f		10	50	ns	Figure 1
DATA STROBE					
PW _{ST}	150ns		10µs		Figure 2
ADDRESS, CHIP SELECT					
Set-up time	125			ns	Figure 2
Hold time	50			ns	Figure 2
DATA BUS—LOADING					
Set-up time	125			ns	Figure 2
Hold time	75			ns	Figure 2
DATA BUS—READING					
T _{OE1}			125	ns	Figure 2, CL = 50pF
T _{DEL1}	5		60	ns	Figure 2, CL = 50pF
OUTPUTS: HQ-7, HS, VS, BL, CRV					
CS-T _{OE1}			125	ns	Figure 1, CL = 20pF
OUTPUTS: RQ-3, DRQ-5					
T _{OE1}	*		500	ns	Figure 3, CL = 20pF

*RQ-3 and DRQ-5 may change prior to the falling edge of H sync

Restrictions

- Only one pin is available for strobing data into the device via the data bus. The cursor X and Y coordinates are therefore loaded into the chip by presenting one set of addresses and outputted by presenting a different set of addresses. Therefore the standard WRITE and READ control signals from most microprocessors must be "NORed" externally to present a single strobe (CS) signal to the device.
- In interlaced mode the total number of character slots assigned to the horizontal scan must be even to insure that vertical sync occurs precisely between horizontal sync pulses.

The figure contains two timing diagrams. The top diagram, labeled 'HORIZONTAL TIMING', shows a horizontal sync pulse followed by a series of horizontal lines. Key parameters labeled include: 'START OF LINE N', 'START OF LINE N+1', 'ACTIVE VIDEO CHARACTERIS PER DATA LINE', 'HORIZONTAL SYNC DELAY (FRONT PORCH)', 'HORIZONTAL SYNC WIDTH', and 'HORIZONTAL LINE COUNT = n'. The bottom diagram, labeled 'VERTICAL TIMING', shows a vertical sync pulse followed by a series of vertical lines. Key parameters labeled include: 'START OF FRAME M (ODD FIELD)', 'START OF FRAME M+1 (EVEN FIELD)', 'SCAN LINES PER FRAME', 'ACTIVE VIDEO DATA LINES PER FRAME', 'VERTICAL DATA', and 'VERTICAL SYNC'.

When employing microprocessor controlled loading of the CRT 5027's registers, the following sequence of instructions is necessary:

ADDRESS	COMMAND
1 1 1 0	Start Timing Chain
1 0 1 0	Reset
0 0 0 0	Load Register 0
.	.
.	.
0 1 1 0	Load Register 6
1 1 1 0	Start Timing Chain

The sequence of START RESET LOAD START is necessary to insure proper initialization of the registers.

This sequence is not required if register loading is via either of the Self Load modes. This sequence is optional with the CRT 5037 or CRT 5057.

STANDARD MICROSYSTEMS
CORPORATION

© 1997 by The McGraw-Hill Companies, Inc.

Circuit diagrams utilizing SMC products are included as a means of illustrating typical semiconductor applications. Immediately complete information sufficient for construction of a circuit is not necessarily given. The information is designed and intended to be used by the engineer or designer. Responsibility for the proper selection for an application and the proper use of the information is assumed by the user. Furthermore, such information does not convey to the purchaser of the semiconductor devices described any license under the patent rights of SMC. SMC reserves the right to make changes at any time in order to improve design and supply the best product possible.

Appendix C9

STANDARD MICROSYSTEMS CORPORATION

15 Morris Blvd., Hawthorne, N.Y. 10881
Tel: 516-331-5001 Fax: 516-331-6000
We keep ahead of our competition so you can keep ahead of yours.

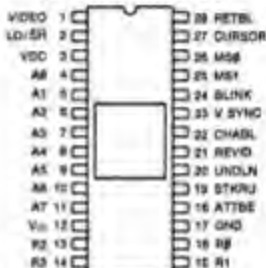
CRT 8002 μPC FAMILY

CRT Video Display Attributes Controller Video Generator VDAC™

FEATURES

- ☐ On chip character generator (mask programmable)
 - 128 Characters (alphanumeric and graphic)
 - 7 x 11 Dot matrix block
- ☐ On chip video shift register
 - Maximum shift register frequency
 - CRT 8002A 20MHz
 - CRT 8002B 15MHz
 - CRT 8002C 10MHz
 - Access time 400ns
- ☐ On chip horizontal and vertical retrace video blanking
- ☐ No descender circuitry required
- ☐ Four modes of operation (intermixable)
 - Internal character generator (ROM)
 - Wide graphics
 - Thin graphics
 - External inputs (font/dot graphics)
- ☐ On chip attribute logic—character, field
 - Reverse video
 - Character blank
 - Character blink
 - Underline
 - Strike-thru
- ☐ Four on chip cursor modes
 - Underline
 - Blinking underline
 - Reverse video
 - Blinking reverse video
- ☐ Programmable character blink rate
- ☐ Programmable cursor blink rate
- ☐ Subscriptable
- ☐ Expandable character set
 - External fonts
 - Alphanumeric and graphic
 - RAM, ROM, and PROM
- ☐ On chip address buffer
- ☐ On chip attribute buffer
- ☐ +5 volt operation
- ☐ TTL compatible
- ☐ MOS N-channel silicon-gate CGPLAMOS® process
- ☐ CLASP® technology—ROM and options
- ☐ Compatible with CRT 5027 VTAC®

PIN CONFIGURATION



General Description

The SMC CRT 8002 Video Display Attributes Controller (VDAC) is an 8-channel CGPLAMOS® MOS-LSI device which utilizes CLASP® technology. It contains a 7X11X28 character generator ROM, a wide graphics mode, a thin graphics mode, an external input mode, character address/data latch, field and/or character attribute logic, attribute latch, four cursor modes, two programmable blink rates, and a high speed video shift register. The CRT 8002 VDAC™ is a companion chip to SMC's CRT 5027 VTAC. Together these two chips comprise the circuitry required for the display portion of a CRT video terminal.

The CRT 8002 video output may be connected directly to a CRT monitor video input. The CRT 5027 blanking output can be connected directly to the CRT 8002 retrace blank input to provide both horizontal and vertical retrace blanking of the video output.

Four cursor modes are available on the CRT 8002. They are: underline, blinking underline, reverse video block, and blinking reverse video block. Any one of these can be mask programmed as the cursor function. There is a separate cursor blink rate which can be mask programmed to provide a 15Hz to 1Hz blink rate.

The CRT 8002 attributes include: reverse video, character blank, blink, underline, and strike-thru. The character blink rate is mask programmable from 7.5Hz to 0.5Hz and has a duty cycle of 75/25. The underline and strike-thru are similar but independently controlled functions and can be mask programmed to any number of raster lines at any position in the character block. These attributes are available in all modes.

In the wide graphic mode the CRT 8002 produces a graphic entity the size of the character block. The graphic entity contains 8 parts, each of which is associated with one bit of a graphic byte, thereby providing for 256 unique graphic symbols. Thus, the CRT 8002 can produce either an alphanumeric symbol or a graphic entity depending on the mode selected. The mode can be changed on a per character basis.

The thin graphic mode enables the user to create single line drawings and forms.

The external mode enables the user to extend the on-chip ROM character set and/or the on-chip graphics capabilities by inserting external symbols. These external symbols can come from either RAM, ROM or PROM.

MAXIMUM GUARANTEED RATINGS*

Operating Temperature Range	-55°C to +70°C
Storage Temperature Range	-55°C to +150°C
Lead Temperature (soldering, 10 sec.)	+325°C
Positive Voltage on any Pin, with respect to ground	+8.0V
Negative Voltage on any Pin, with respect to ground	-0.3V

*Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

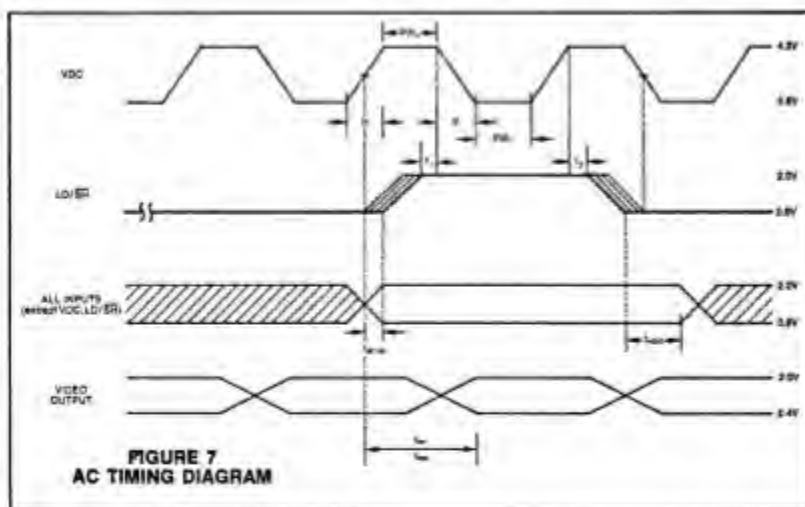
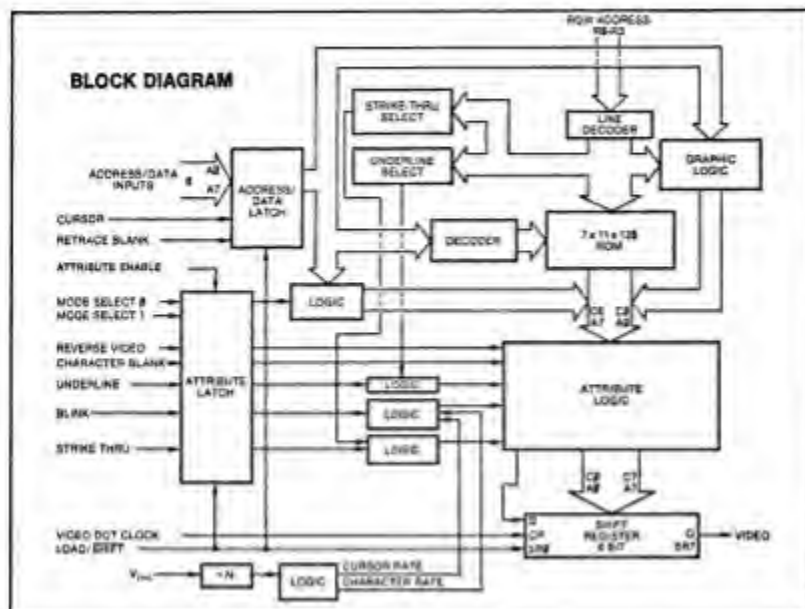
NOTE: When powering this device from laboratory or system power supplies, it is important that the Absolute Maximum Ratings not be exceeded or device failure can result. Some power supplies exhibit voltage spikes or "glitches" on their outputs when the AC power is switched on and off. In addition, voltage transients on the AC power line may appear on the DC output. If this possibility exists it is suggested that a clamp circuit be used.

ELECTRICAL CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5% unless otherwise noted)

Parameter	Min.	Typ.	Max.	Unit	Comments
D.C. CHARACTERISTICS					
INPUT VOLTAGE LEVELS					
Low-level, V_{IL}	2.0		0.8	V	excluding VDC
High-level, V_{IH}				V	excluding VDC
INPUT VOLTAGE LEVELS—CLOCK					
Low-level, V_{IL}	4.3		0.8	V	See Figure 6
High-level, V_{IH}				V	
OUTPUT VOLTAGE LEVELS					
Low-level, V_{OL}	2.4		0.4	V	$I_{OL} = 0.4 \text{ mA}$, 74LSXX load
High-level, V_{OH}				V	$I_{OH} = -20 \mu\text{A}$
INPUT CURRENT					
Leakage, I_i (Except CLOCK)			10	μA	$0 \leq V_{IL} \leq V_{IH}$
Leakage, I_i (CLOCK Only)			50	μA	$0 \leq V_{IL} \leq V_{IH}$
INPUT CAPACITANCE					
Data		10		pF	① 1 MHz
LD/SR		20		pF	② 1 MHz
CLOCK		25		pF	③ 1 MHz
POWER SUPPLY CURRENT					
I_{CC}		100		mA	
A.C. CHARACTERISTICS					
See Figure 6, 7					

PRELIMINARY

SYMBOL	PARAMETER	CRT 8002A		CRT 8002B		CRT 8002C		UNITS
		MIN.	MAX.	MIN.	MAX.	MIN.	MAX.	
V _{DC}	Video Dot Clock Frequency	1.0	20	1.0	15	1.0	10	MHz
PW _H	V _{DC} —High Time	15.0		23		40		ns
PW _L	V _{DC} —Low Time	15.0		23		40		ns
t _{cy}	LD/SR cycle time	400		533		800		ns
t _r , t _f	Rise, fall time		10		10		10	ns
t _{set-up}	Input set-up time	≥ 0		≥ 0		≥ 0		ns
t _{hold}	Input hold time	15		15		15		ns
t _{ex} , t _{sed}	Output propagation delay	15	50	15	55	15	100	ns
t ₁	LD/SR set-up time	10		15		20		ns
t ₂	LD/SR hold time	15		15		15		ns



DESCRIPTION OF PIN FUNCTIONS

PIN NO.	SYMBOL	NAME	INPUT/ OUTPUT	FUNCTION
1	VIDEO	Video Output	O	The video output contains the dot stream for the selected row of the alphanumeric, wide graphic, thin graphic, or external character after processing by the attribute logic and the retrace blank and cursor inputs. In the alphanumeric mode, the characters are ROM programmed into the 77 dots (TX11) allocated for each of the 128 characters. See figure 5. The top row (R0) and rows R12 to R15 are normally all zeros as is column C7. Thus, the character is defined in the box bounded by R1 to R11 and C0 to C6. When a row of the ROM, via the attribute logic, is parallel loaded into the 8-bit shift register, the first bit serially shifted out is C7 (A zero or a one in REVID). It is followed by C6, C5, through C0. The timing of the Load/Shift pulse will determine the number of additional (n - 1) zeros to be shifted out (or ones if in REVID) shifted out. See figure 4. When the next Load/Shift pulse occurs the next character's row of the ROM, via the attribute logic, is parallel loaded into the shift register and the cycle repeats.
2	LD/SH	Load/Shift	I	The 8 bit shift-register parallel-in load or serial-out shift modes are established by the Load/Shift input. When low, this input enables the shift register for serial shifting with each Video Out Clock pulse. When high, the shift register serial (broadcast) data inputs are enabled and synchronous loading occurs on the next Video Out Clock pulse. During parallel loading, serial data flow is inhibited. The Address/Data inputs (A0-A7) are latched on the negative transition of the Load/Shift input. See timing diagram, figure 7.
3	VOC	Video Out Clock	I	Frequency at which video is shifted.
4-11	A0-A7	Address/Data	I	In the Alphanumeric Mode the 7 bits on inputs (A0-A6) are internally decoded to address one of the 128 available characters (A7 = K). In the External Mode, A0-A7 is used to insert an 8 bit word from a user defined external ROM, PROM, or RAM into the on-chip Attribute Logic. In the wide Graphic Modes A0-A7 is used to define one of 256 graphic entities. In the Thin Graphic Mode A0-A2 is used to define the 2 line segments.
12	V _{cc}	Power Supply	PS	+5 volt power supply
13, 14, 15, 16, 17, 18, 19, 20	R0-R15	Row Address	I	These 4 binary inputs define the row address in the current character block.
17	GND	Ground	GND	Ground
18	ATTBE	Attribute Enable	I	A positive level on this input enables data from the Reverse Video, Character Blank, Underline, Strike-Thru, Blink, Mode Select 0, and Mode Select 1 inputs to be loaded into the on-chip attribute latch at the negative transition of the Load/Shift pulse. The latch loading is disabled when this input is low. The latched attributes will remain fixed until this input becomes high again. To facilitate attribute latching on a character by character basis, see ATTBE high. See timing diagram, figure 7.
19	STRIK	Strike-Thru	I	When this input is high and RETBL = 0, the parallel inputs to the shift register are forced high (S0-SRT), providing a solid line segment throughout the character block. The operation of strike-thru is modified by Reverse Video (see table 1). In addition, an on-chip ROM programmable decoder is available to decide the line count on which strike-thru is to be placed as well as to program the strike-thru to be 1 to N raster lines high. Actually, the strike-thru decoder (mask programmable) logic allows the strike-thru to be any number or arrangement of horizontal lines in the character block. The standard strike-thru will be a double line on rows R5 and R6.
20	UNDRN	Underline	I	When this input is high and RETBL = 0, the parallel inputs to the shift register are forced high (S0-SRT), providing a solid line segment throughout the character block. The operation of underline is modified by Reverse Video (see table 1). In addition, an on-chip ROM programmable decoder is available to decide the line count on which underline is to be placed as well as to program the underline to be 1 to N raster lines high. Actually, the underline decoder (mask programmable) logic allows the underline to be any number or arrangement of horizontal lines in the character block. The standard underline will be a single line on R11.
21	REVID	Reverse Video	I	When this input is low and RETBL = 0, data into the Attribute Logic is presented directly to the shift register parallel inputs. When reverse video is high data into the Attribute Logic is reversed and then presented to the shift register parallel inputs. This operation reverses the data and field video. See table 1.
22	CHABL	Character Blank	I	When this input is high, the parallel inputs to the shift register are all set low, providing a blank character line segment. Character blank will override blink. The operation of Character Blank is modified by the Reverse Video input. See table 1.
23	V SYNC	V SYNC	I	This input is used as the clock input for the two on-chip mask programmable blink rate dividers. The cursor blink rate (50/50 duty cycle) will be twice the character blink rate (75/25 duty cycle). The dividers can be programmed from - 4 to - 32 for the cursor (- 5 to - 80 for the character).
24	BLINK	Blink	I	When this input is high and RETBL = 0 and CHABL = 0, the character will blink at the programmed character blink rate. Blinking is accomplished by blanking the character block with the internal Character Blink clock. The standard character blink rate is 1.875 Hz.
25	MS1	Mode Select 1	I	These 2 inputs define the four modes of operation of the GRT 8002 as follows: Alphanumeric Mode - In this mode addresses A0-A6 (A7 = K) are internally decoded to address 1 of the 128 available ROM characters. The addressed character along with the decoded row will define a 7 bit output from the ROM to be loaded into the shift register via the attribute logic. Thin Graphic Mode - In this mode A0-A2 (A3-A7 = K) will be loaded into the thin graphic logic along with the row addresses. This logic will define the segments of a graphic entity as defined in figure 2. The top of the entity will begin on row 0000 and will end on a mask programmable row.
26	MS0	Mode Select 0	I	
	MS1	MS0	MODE	
	1	1	Alphanumeric	
	1	0	Thin Graphic	
	0	1	External Mode	
	0	0	Wide Graphic	

DESCRIPTION OF PIN FUNCTIONS

PIN NO.	SYMBOL	NAME	INPUT/OUTPUT	FUNCTION
25 26 (cont.)				<p>External Mode—In this mode the inputs A2-A7 go directly from the character latch into the shift register via the attribute logic. Thus the user may define external character fonts or graphic entities in an external PROM, ROM or RAM. See figure 3.</p> <p>Wide Graphics Mode—In this mode the inputs A2-A7 will define a graphic entity as described in figure 1. Each line of the graphic entity is determined by the wide graphic logic in conjunction with the row inputs R6 to R3. In this mode each segment of the entity is defined by one of the bits of the 8-bit word. Therefore, the 8 bits can define any 1 of the 256 possible graphic entities. These entities can be put up against each other to form a contiguous pattern or can be interspersed with alphanumeric characters. Each of the entities occupies the space of 1 character block and thus requires 1 byte of memory.</p> <p>These 4 modes can be interleaved on a per character basis.</p>
27	CURSOR	Cursor	1	<p>When this input is enabled 1 of the 4 pre-programmed cursor modes will be activated. The cursor mode is on-chip mask programmable. The standard cursor will be a blinking (at 3.75 Hz) reverse video block. The 4 cursor modes are:</p> <p>Underline—In this mode an underline (1 to N raster lines) as the programmed underline position occurs.</p> <p>Blinking Underline—In this mode the underline blinks at the cursor rate.</p> <p>Reverse Video Block—In this mode the Character Block is set to reverse video.</p> <p>Blinking Reverse Video Block—In this mode the Character Block is set to reverse video at the cursor blink rate. The Character Block will alternate between normal video and reverse video.</p> <p>The cursor functions are listed in table 1.</p>
28	RETEL	Replace Blank	1	<p>When this input is latched high, the shift register parallel inputs are unconditionally cleared to all zeros and loaded into the shift register on the next Load/Shift pulse. This blanks the video, independent of all attributes, during horizontal and vertical retrace time.</p>

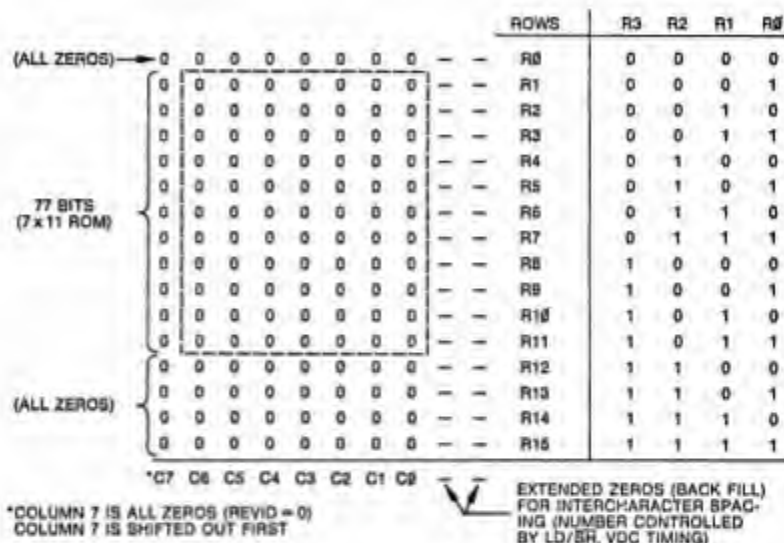
TABLE 1

CURSOR	RETEL	REVID	CHABL	UNOLN*	FUNCTION
X	1	X	X	X	"0" (S.R.) All
0	0	0	0	0	"D" (S.R.) All
0	0	0	0	1	"1" (S.R.) *
0	0	0	1	X	"0" (S.R.) All others
0	0	1	0	0	"D" (S.R.) All
0	0	1	0	1	"1" (S.R.) *
0	0	1	1	X	"0" (S.R.) All others
Underline*	0	0	0	X	"1" (S.R.) *
Underline*	0	0	1	X	"D" (S.R.) All others
Underline*	0	1	0	X	"1" (S.R.) *
Underline*	0	1	1	X	"D" (S.R.) All others
Underline*	0	1	1	X	"0" (S.R.) *
Blinking** Underline*	0	0	0	X	"1" (S.R.) * Blinking
Blinking** Underline*	0	0	1	X	"D" (S.R.) All others
Blinking** Underline*	0	1	0	X	"1" (S.R.) * Blinking
Blinking** Underline*	0	1	1	X	"D" (S.R.) All others
Blinking** Underline*	0	1	1	X	"0" (S.R.) * Blinking
REVID Block	0	0	0	0	"D" (S.R.) All
REVID Block	0	0	0	1	"D" (S.R.) *
REVID Block	0	0	1	X	"1" (S.R.) All others
REVID Block	0	0	0	1	"0" (S.R.) *
REVID Block	0	1	0	0	"D" (S.R.) All others
REVID Block	0	1	0	1	"D" (S.R.) All
REVID Block	0	1	1	X	"1" (S.R.) *
REVID Block	0	1	1	X	"D" (S.R.) All others
Blink** REVID Block	0	0	0	0	"D" (S.R.) All
Blink** REVID Block	0	0	0	1	"D" (S.R.) *
Blink** REVID Block	0	0	1	X	"1" (S.R.) All others
Blink** REVID Block	0	1	0	0	"D" (S.R.) All
Blink** REVID Block	0	1	0	1	"D" (S.R.) All
Blink** REVID Block	0	1	1	X	"1" (S.R.) *

*At Selected Row Decode **At Cursor Blink Rate

Note: If Character is Blinking at Character Rate, Cursor will change it to Cursor Blink Rate.

FIGURE 5
ROM CHARACTER BLOCK FORMAT



HEX	ASCII	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	0	N	A	S	E	F	F	A	E	B	4	F	Y	F	R	S	S
01	1	R	9	5	B	4	K	7	5	W	5	5	5	5	5	5	5
02	2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
03	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
04	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
05	5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
06	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
07	7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	%

CONSULT FACTORY FOR CUSTOM FONT AND OPTION PROGRAMMING FORMS.

FIGURE 1
WIDE GRAPHICS MODE
MSB=0 MS1=0

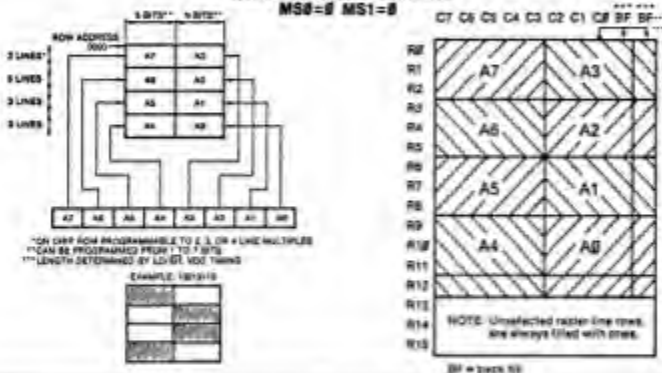


FIGURE 2
THIN GRAPHICS MODE
MSB=0 MS1=1

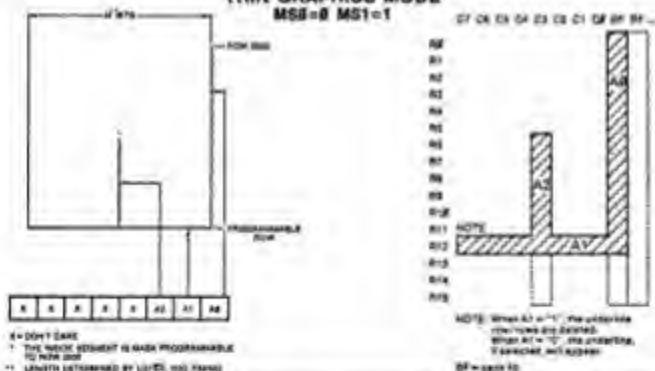


FIGURE 3
EXTERNAL MODE
MSB=1 MS1=0

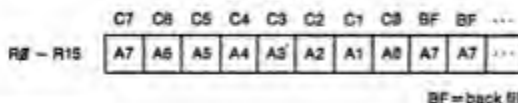


FIGURE 4 TYPICAL VIDEO OUTPUT

The diagram shows the timing of video output signals. The top signal is VDC, which is a periodic square wave. Below it is the LD/ST signal, which is high during the first two dot fields and then low. The video data is shown for two fields: 8 DOT FIELD and 9 DOT FIELD. Each field is divided into 16 horizontal lines, each containing 16 characters. The characters are arranged in a grid. A legend indicates that 'C' is the character number and 'y' is the column number. A box labeled 'Alphanumeric' and 'External' is shown, indicating the type of data being output. The signal is labeled 'BF=back fill'.

NOTE: C = character number
y = column number

Alphanumeric
External

BF=back fill

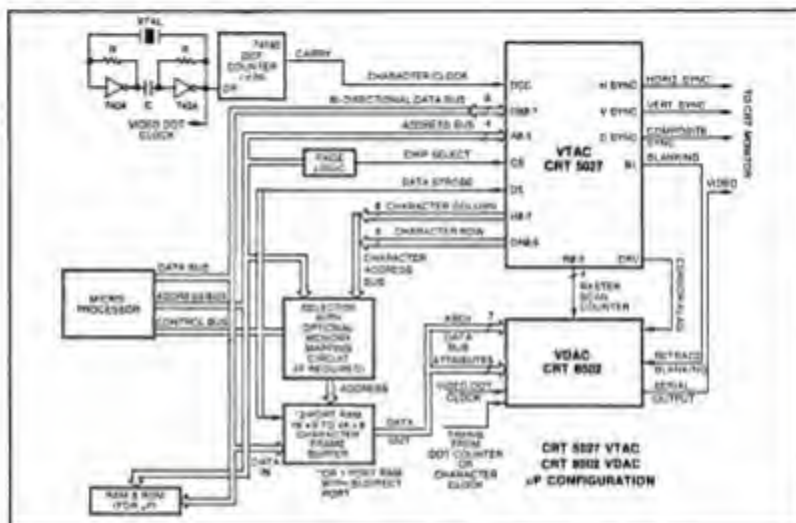


FIGURE 6

The diagram shows a circuit for the 74S74 flip-flop. The external clock signal $CP_{EXTERNAL}$ is connected to the input of an inverter. The output of the inverter is connected to the CLK input of the 74S74 flip-flop. A 500Ω resistor is connected between the output of the inverter and the V_{CC} supply. The output of the flip-flop, Q , is connected to the LD/SR output. The external load/shift signal $LOAD/SHIFT_{EXTERNAL}$ is connected to the D input of the flip-flop. The output of the flip-flop is also connected to the V_{CC} supply.

© 1999 by The McGraw-Hill Companies, Inc. All rights reserved. Printed in the United States of America. This publication is protected by copyright. Any unauthorized distribution or reproduction of this work is prohibited. For more information, contact The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020-1095.

Circuit diagrams using SMC products are included as a means of illustrating typical semiconductor applications. Consequently, circuit diagrams are not intended to be construed as a design or construction guide. The user assumes all responsibility for safety and reliability of the circuit. The user is advised that the information is not intended for use in the design of life-critical systems. Furthermore, such information does not convey to the purchaser of the semiconductor the exact semiconductor device or device under the patent rights of SMC or others. SMC reserves the right to make changes at any time in order to improve design and supply the best product possible.

Appendix C10

STANDARD MICROSYSTEMS CORPORATION
© Microware Corp., Providence 02917, U.S.A.
 508-273-0500 1989-2000 Microware
 We keep ahead of our competition so you can keep ahead of yours.

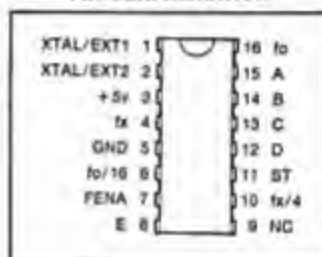
**COM 8046
COM 8046T**

Baud Rate Generator Programmable Divider

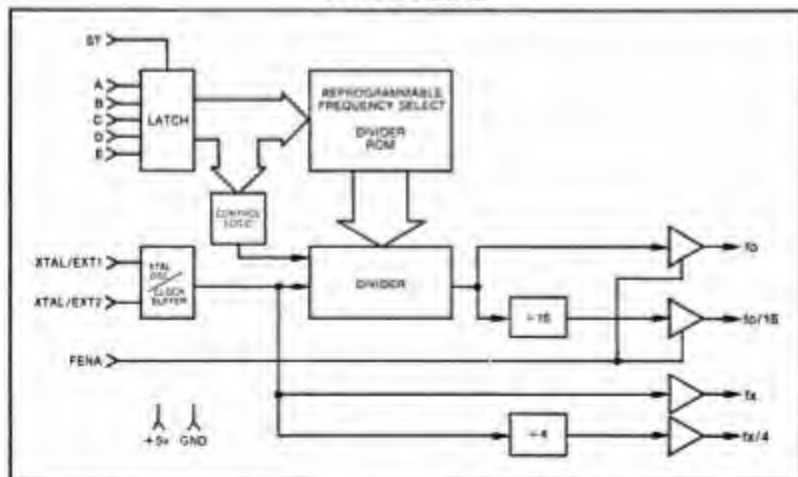
FEATURES

- ☐ On chip crystal oscillator or external frequency input
- ☐ Single +5v power supply
- ☐ Choice of 32 output frequencies
- ☐ 32 asynchronous/synchronous baud rates
- ☐ Direct UART/USRT/ASTRO/USYNRT compatibility
- ☐ Re-programmable ROM via CLASP® technology allows generation of other frequencies
- ☐ TTL, MOS compatible
- ☐ 1X Clock via fo/16 output
- ☐ Crystal frequency output via fx and fx/4 outputs
- ☐ Output disable via FENA

PIN CONFIGURATION



BLOCK DIAGRAM



General Description

The Standard Microsystems COM 8046 is an enhanced version of the COM 8045 Baud Rate Generator. It is fabricated using SMC's patented COPLANOS[®] and CLASP[®] technologies and employs depletion mode loads, allowing operation from a single +5v supply.

The standard COM 8046 is specifically dedicated to generating the full spectrum of 16 asynchronous/asynchronous data communication frequencies for 1X, 16X and 32X UART/USRT/ASTRO/USYHRT devices.

The COM 8046 features an internal crystal oscillator which may be used to provide the master reference frequency. Alternatively, an external reference may be supplied by applying complementary TTL level signals to pins 1 and 2. Parts suitable for use only with an external TTL reference are marked COM 8046T. TTL outputs used to drive the COM 8046 or COM 8046T should not be used to drive other TTL inputs, as noise immunity may be compromised due to excessive loading.

The reference frequency (f_x) is used to provide two high frequency outputs: one at f_x and the other at $f_x/4$. The $f_x/4$ output will drive one standard 7400 load, while the f_x output will drive two 74LS loads.

The output of the oscillator/buffer is applied to the divider for generation of the output frequency f_o . The divider is capable of dividing by any integer from 8

to $2^n + 1$, inclusive. If the divisor is even, the output will be square; otherwise the output will be high longer than it is low by one f_x clock period. The output of the divider is also divided internally by 16 and made available at the $f_o/16$ output pin. The $f_o/16$ output will drive one and the f_o output will drive two standard 7400 TTL loads. Both the f_o and $f_o/16$ outputs can be disabled by supplying a low logic level to the FENA input pin. Note that the FENA input has an internal pull-up which will cause the pin to rise to approximately V_{cc} if left unconnected.

The divisor ROM contains 32 divisors, each 19 bits wide, and is fabricated using SMC's unique CLASP[®] technology. This process permits reduction of turn-around-time for ROM patterns.

The five divisor select bits are held in an externally strobed data latch. The strobe input is level sensitive: while the strobe is high, data is passed directly through to the ROM. Initiation of a new frequency is effected within 3.5 μ s of a change in any of the five divisor select bits; strobe activity is not required. This feature may be disabled through a CLASP[®] programming option causing new frequency initiation to be delayed until the end of the current f_o half-cycle. All five data inputs have pull-ups identical to that of the FENA input, while the strobe input has no pull-up.

Description of Pin Functions

Pin No.	Symbol	Name	Function
1	XTAL/EXT1	Crystal or External Input 1	This input is either one pin of the crystal package or one polarity of the external input.
2	XTAL/EXT2	Crystal or External Input 2	This input is either the other pin of the crystal package or the other polarity of the external input.
3	V_{cc}	Power Supply	+5 volt supply
4	f_o	f_o	Crystal/clock frequency reference output
5	GND	Ground	Ground
6	$f_o/16$	$f_o/16$	1X clock output
7	FENA	Enable	A low level at this input causes the f_o and $f_o/16$ outputs to be held high. An open or a high level at the FENA input enables the f_o and $f_o/16$ outputs.
8	E	E	Most significant divisor select data bit. An open at this input is equivalent to a logic high.
9	NC	NC	No connection
10	$f_x/4$	$f_x/4$	1/4 crystal/clock frequency reference output.
11	ST	Strobe	Divisor select data strobe. Data is sampled when this input is high, preserved when this input is low.
12-15	D,C,B,A	D,C,B,A	Divisor select data bits. A=LSB. An open circuit at these inputs is equivalent to a logic high.
16	f_x	f_x	16X clock output

ELECTRICAL CHARACTERISTICS COM8046, COM8046T, COM8116, COM8116T, COM8126, COM8126T, COM8136, COM8136T, COM8146, COM8146T

MAXIMUM GUARANTEED RATINGS*

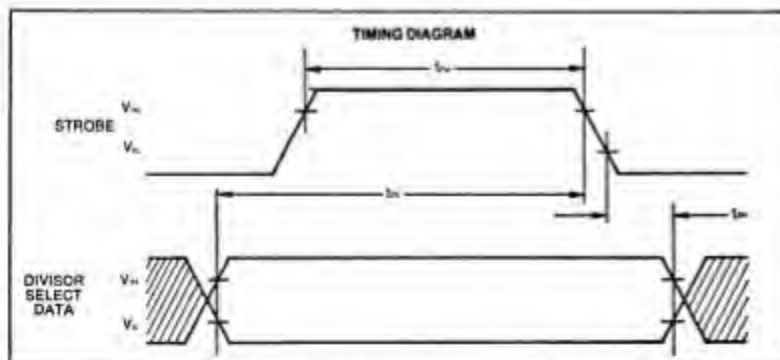
Operating Temperature Range	-55°C to +70°C
Storage Temperature Range	-55°C to +150°C
Lead Temperature (soldering, 10 sec.)	+325°C
Positive Voltage on any Pin, with respect to ground	+5.0V
Negative Voltage on any Pin, with respect to ground	-0.3V

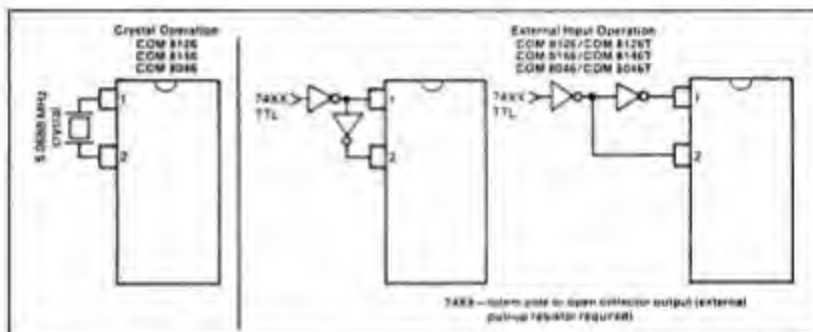
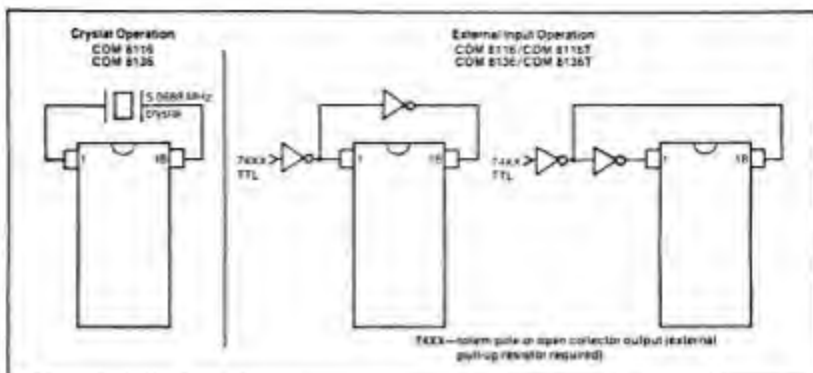
*Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied.

NOTE: When powering this device from laboratory or system power supplies, it is important that the Absolute Maximum Ratings not be exceeded or device failure can result. Some power supplies exhibit voltage spikes or "glitches" on their outputs, when the AC power is switched on and off. In addition, voltage transients on the AC power line may appear on the DC output. If this possibility exists it is suggested that a clamp circuit be used.

ELECTRICAL CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ± 5%, unless otherwise noted)

Parameter	Min.	Typ.	Max.	Unit	Comments
D.C. CHARACTERISTICS					
INPUT VOLTAGE LEVELS					
Low-level, V _{IL}	2.0		0.8	V	excluding XTAL inputs
High-level, V _{IH}				V	
OUTPUT VOLTAGE LEVELS					
Low-level, V _{OL}			0.4	V	I _{OL} = 1.6mA, for I _{OL} /4, I _{OL} /16
			0.4	V	I _{OL} = 3.2mA, for I _{OL} , I _{OL}
			0.4	V	I _{OL} = 0.8mA, for I _{OL}
High-level, V _{OH}	3.5			V	I _{OH} = -100μA for I _{OH} , I _{OH} = -50μA
INPUT CURRENT					
Low-level, I _{IL}			-0.1	mA	V _{IL} = GND, excluding XTAL inputs
INPUT CAPACITANCE					
All inputs, C _i	8	10		pF	V _{IL} = GND, excluding XTAL inputs
EXT. INPUT LOAD					
Series 7400 equivalent loads	8	10			
POWER SUPPLY CURRENT					
I _{CC}			50	mA	
A.C. CHARACTERISTICS					
CLOCK FREQUENCY, f_{CLK}					
	0.01		7.0	MHz	T _A = +25°C
	0.01		5.1	MHz	XTAL/EXT, 50% Duty Cycle ±5%
					COM 8046, COM 8126, COM 8146
					XTAL/EXT, 90% Duty Cycle ±5%
					COM 8116, COM 8136
STROBE PULSE WIDTH, t_{SP}					
	150		DC	ns	
INPUT SET-UP TIME					
t _{su}	200			ns	
INPUT HOLD TIME					
t _{sh}	50			ns	
STROBE TO NEW FREQUENCY DELAY					
			3.5	μs	@ f _{CLK} = 5.0 MHz





For ROM re-programming SMC has a computer program available whereby the customer need only supply the input frequency and the desired output frequencies. The ROM programming is automatically generated.

Crystal Specifications

User must specify termination (pin wire short)
Prefer HC-18/U or HC-25/U
Frequency — 5.0668 MHz AT cut
Temperature range 0°C to 70°C
Series resistance: 30 Ω
Series Resonant
Overall tolerance: ± 0.1%,
or as required

Crystal manufacturers (Please List)

Northern Engineering Laboratories
357 Beloit Street
Burlington, Wisconsin 53105
(414) 763-3591

Bulova Frequency Control Products
61-20 Woodside Avenue
Woodside, New York 11377
(212) 335-6000

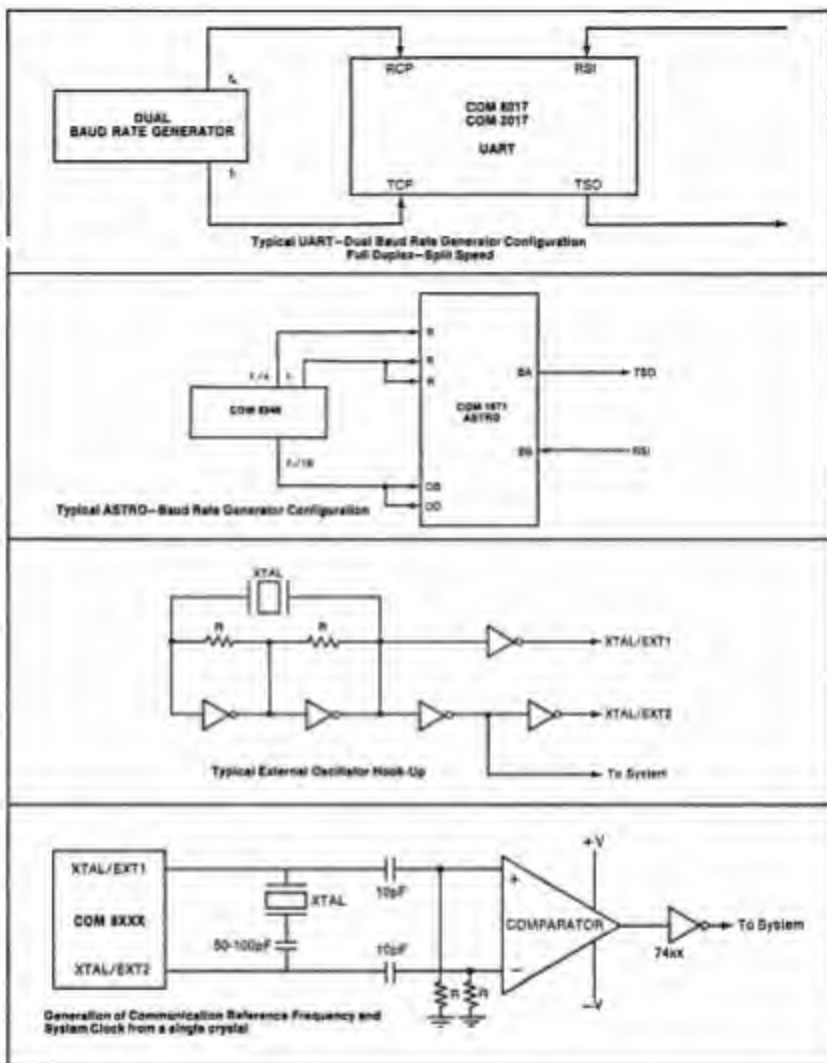
CTS Knights Inc.
101 East Church Street
Sandwich, Illinois 60548
(815) 786-8411

Crytek Crystals Corporation
1000 Crystal Drive
Fort Myers, Florida 33901
(813) 906-2109

COM 8046 COM 8046T

Table 2
REFERENCE FREQUENCY = 5.000000MHz

Divisor Select EDCSA	Desired Band Rate	Clock Factor	Desired Frequency (MHz)	Divisor	Actual Band Rate	Actual Frequency (MHz)	Deviation
00000	50.00	32X	1.60000	3168	50.00	1.600000	0.0000%
00001	75.00	32X	2.40000	2112	75.00	2.400000	0.0000%
00010	110.00	32X	3.52000	1440	110.00	3.520000	0.0000%
00011	133.33	32X	4.33333	1177	133.33	4.333333	0.0001%
00100	150.00	32X	4.80000	1056	150.00	4.800000	0.0000%
00101	200.00	32X	6.40000	792	200.00	6.400000	0.0000%
00110	200.00	32X	6.40000	528	300.00	9.600000	0.0000%
00111	600.00	32X	19.20000	264	600.00	19.200000	0.0000%
01000	1200.00	32X	38.40000	132	1200.00	38.400000	0.0000%
01001	1800.00	32X	57.60000	88	1800.00	57.600000	0.0000%
01010	2400.00	32X	76.80000	66	2400.00	76.800000	0.0000%
01011	3600.00	32X	115.20000	44	3600.00	115.200000	0.0000%
01100	4800.00	32X	153.60000	33	4800.00	153.600000	0.0000%
01101	7200.00	32X	230.40000	22	7200.00	230.400000	0.0000%
01110	9600.00	32X	307.20000	16	9600.00	316.800000	3.1250%
01111	19200.00	32X	614.40000	8	19200.00	633.600000	3.1250%
10000	50.00	16X	0.80000	6336	50.00	0.800000	0.0000%
10001	75.00	16X	1.20000	4224	75.00	1.200000	0.0000%
10010	110.00	16X	1.76000	2880	110.00	1.760000	0.0000%
10011	133.33	16X	2.13333	2352	133.33	2.133333	0.0000%
10100	150.00	16X	2.40000	2112	150.00	2.400000	0.0000%
10101	300.00	16X	4.80000	1056	300.00	4.800000	0.0000%
10110	600.00	16X	9.60000	528	600.00	9.600000	0.0000%
10111	1200.00	16X	19.20000	264	1200.00	19.200000	0.0000%
11000	1800.00	16X	28.80000	176	1800.00	28.800000	0.0000%
11001	2000.00	16X	32.00000	156	2000.00	32.000000	0.0000%
11010	2400.00	16X	38.40000	132	2400.00	38.400000	0.0000%
11011	3600.00	16X	57.60000	88	3600.00	57.600000	0.0000%
11100	4800.00	16X	76.80000	66	4800.00	76.800000	0.0000%
11101	7200.00	16X	115.20000	44	7200.00	115.200000	0.0000%
11110	9600.00	16X	153.60000	33	9600.00	153.600000	0.0000%
11111	19200.00	16X	307.20000	16	19200.00	316.800000	3.1250%



Appendix D ZAP Operating System

Appendix D

ZAP Operating System

FILE 3000 7323
READY
ASST

```

7324      0100 *
7324      0110 *
7324      0120 *THE FOLLOWING EQUATES ARE USED
7324      0130 *OS OPERATING SYSTEM CONSTANTS
7324      0140 *
7324      0150 ZERO EQU 0
7324      0160 ONE EQU 1
7324      0170 TWO EQU 2
7324      0180 THREE EQU 3
7324      0190 FOUR EQU 4
7324      0200 FIVE EQU 5
7324      0210 EIGHT EQU 8
7324      0220 ADDI01 EQU 5 *XADS ADDRESS DISPLAY PORT
7324      0230 ADDI02 EQU 6 *XADS ADDRESS DISPLAY PORT
7324      0240 DATDIS EQU 7 *DATA DISPLAY PORT
7324      0250 EXECC EQU 16H *EXEC KEY
7324      0260 HEXTC EQU 32H *NEXT KEY
7324      0270 UARTIO EQU 2 *UART I/O PORT
7324      0280 UARTST EQU 3 *UART STATUS PORT
7324      0290 KEYPT EQU 0 *KEYBOARD INPUT PORT
7324      0300 *
7324      0310 *
0000      0320 ST 0
0000      0330 *
0000      0340 *
0000      0350 *COLD SETS THE OPERATING SYSTEM STACK POINTER
0000      0360 *AND ENTERS THE COMMAND RECOGNITION MODULE
0000      0370 *
0000      0380 *
0000 31 C4 07 0390 COLD LD SP*SPSTRT *INITIALIZE STACK POINTER
0003 C3 40 00 0400 JP WARM01
0006 0410 DS 2
0009 C3 47 00 0420 WARM JP WARM1 *RST 1 OR WARM START
000B 0430 DS 5
0010 C3 C5 07 0440 RST2E JP RST2H *RST 2 TRANSFER
0013 0450 DS 5
001B C3 C8 07 0460 RST3E JP RST3H *RST 3 TRANSFER
001E 0470 DS 5
0020 C3 C8 07 0480 RST4E JP RST4H *RST 4 TRANSFER
0023 0490 DS 5
002B C3 CE 07 0500 RST5E JP RST5H *RST 5 TRANSFER
002E 0510 DS 5
0030 C3 D1 07 0520 RST6E JP RST6H *RST 6 TRANSFER
0033 0530 DS 5
003B C3 D4 07 0540 RST7E JP RST7H *RST 7 TRANSFER
003E 0550 DS 5
0040 ED 73 DB 07 0551 WARM01 LD (SPLSAV)*SP

```

```

0044 C3 89 00      0552      JP      WARM2      *GO TO COMMAND RECOGNITION
0047      0560 *
0047      0570 *
0047      0580 *
0047      0590 *WARM START SAVES THE USERS REGISTERS AND
0047      0600 *ENTERS THE COMMAND RECOGNITION MODE WITH
0047      0610 *FS DISPLAYED ON THE DATA AND ADDRESS DISPLAYS
0047      0620 *
0047 32 E3 07      0630 WARM1 LD (ASAV),A      *SAVE USERS A
004A E1      0640 POP HL      *GET USERS PC FROM STACK
004B 22 DD 07      0650 LD (PCLSAV),HL *SAVE USERS PC IN SAVE AREA
004E F5      0660 PUSH AF
004F E1      0670 POP HL      *GET USERS FLAGS
0050 22 E7 07      0680 LD (ESAV),HL *SAVE USERS FLAGS
0053 DD 22 D7 07      0690 LD (IXLSAV),IX *SAVE USERS IX
0057 FD 22 D9 07      0700 LD (IYLSAV),IY *SAVE USERS IY
005B ED 73 DB 07      0710 LD (SFLSAV),SP *SAVE USERS SP
005F ED 57      0720 LD A,I      *SAVE USERS I
0061 32 DF 07      0730 LD (ISAV),A
0064 ED 5F      0740 LD A,R      *SAVE USERS R
0066 32 E0 07      0750 LD (RSV),A
0069 21 E4 07      0760 LD HL,BSAV
006C 70      0770 LD (HL),B      *SAVE USERS B
006D 23      0780 INC HL
006E 71      0790 LD (HL),C      *SAVE USERS C
006F 23      0800 INC HL
0070 72      0810 LD (HL),D      *SAVE USERS D
0071 23      0820 INC HL
0072 73      0830 LD (HL),E      *SAVE USERS E
0073 08      0840 EX AF,AF      *SAVE ALTERNATE REGISTERS
0074 F5      0850 PUSH AF
0075 32 EB 07      0860 LD (AASAV),A      *SAVE ALT A
007B 22 E9 07      0870 LD (ALSAV),HL      *SAVE ALT H&L
007B E1      0880 POP HL
007C 22 EF 07      0890 LD (AESAV),HL      *SAVE ALT FLAGS
007F 21 EC 07      0900 LD HL,ABS&V
0082 70      0910 LD (HL),B      *SAVE ALT B
0083 23      0920 INC HL
0084 71      0930 LD (HL),C      *SAVE ALT C
0085 23      0940 INC HL
0086 72      0950 LD (HL),D      *SAVE ALT D
0087 23      0960 INC HL
0088 73      0970 LD (HL),E      *SAVE ALT E
0089      0980 *
0089      0990 *
0089      1000 *COMMAND RECOGNITION MODULE
0089      1010 *
0089 CD F1 00      1020 WARM2 CALL CLOIS      *CLEAR DISPLAY
008C 3E FF      1030 LD A,255D      *DISPLAY FFFF FF
008E D3 05      1040 OUT ADDIS1
0090 D3 06      1050 OUT ADDIS2
0092 D3 07      1060 OUT DATDIS
0094 CD 03 01      1070 CALL KEYIN      *GET INPUT CHARACTER
0097 06 40      1080 LD B,KEYH
0099 B8      1090 CP B
009A CA F1 01      1100 JP Z,MEMORY      *JUMP IF MEMORY REQUEST
009D 04      1110 INC B
009E B8      1120 CP B
009F CA 4B 02      1130 JP Z,REGIST      *JUMP IF REGISTER REQUEST
00A2 04      1140 INC B
00A3 B8      1150 CP B
00A4 CA 10 03      1160 JP Z,GOREQ
00A7 C3 89 00      1170 JP WARM2
00AA      1180 *
00AA      1190 MEM EQU 64D      *MEMORY KEY
00AA      1200 *
00AA      1210 *
00AA      1220 *RESTART RESTORES THE USERS REGISTERS
00AA      1230 *AND RETURNS CONTROL TO THE ADDRESS

```

00AA		1240	*SPECIFIED IN THE PC SAVE LOCATION IN THE		
00AA		1250	*REGISTER SAVE AREA		
00AA		1260	*		
00AA 3A EC 07	1270	RESTRT	LD	A,(ABSAV)	*RESTORE ALT REGISTERS
00AD 47	1280		LD	B,A	
00AE 3A ED 07	1290		LD	A,(ACSAV)	
00B1 AF	1300		LD	C,A	
00B2 3A EF 07	1310		LD	A,(ADSAV)	
00B5 57	1320		LD	D,A	
00B6 3A EF 07	1330		LD	A,(AESAV)	
00B9 5F	1340		LD	E,A	
00BA 3A F0 07	1350		LD	A,(AFSAV)	
00BD 6F	1360		LD	L,A	
00BE E5	1370		PUSH	HL	
00BF F1	1380		POP	AF	
00C0 3A FB 07	1390		LD	A,(AASAV)	
00C3 2A E9 07	1400		LD	HL,(ALSAV)	
00C6 D9	1410		EXX		
00C7 FD 2A D9 07	1420		LD	IY,(IYLSAV)	*RESTORE IY
00CB DD 2A D7 07	1430		LD	IX,(IXLSAV)	*RESTORE IX
00CF 21 DF 07	1440		LD	HL,ISAV	
00D2 7E	1450		LD	A,(HL)	
00D3 ED 47	1460		LD	I,A	
00D5 23	1470		INC	HL	
00D6 7E	1480		LD	A,(HL)	
00D7 ED 4F	1490		LD	R,A	
00D9 21 E3 07	1500		LD	HL,ASAV	
00DC 7E	1510		LD	A,(HL)	*RESTORE A
00DB 23	1520		INC	HL	
00DE 46	1530		LD	B,(HL)	*RESTORE B
00DF 23	1540		INC	HL	
00E0 4E	1550		LD	C,(HL)	*RESTORE C
00E1 23	1560		INC	HL	
00E2 56	1570		LD	D,(HL)	*RESTORE D
00E3 23	1580		INC	HL	
00E4 5E	1590		LD	E,(HL)	*RESTORE E
00E5 ED 7B DB 07	1600		LD	SP,(SPLSAV)	*RESTORE STACK POINTER
00E9 2A DD 07	1610		LD	HL,(PCLSAV)	*REPLACE PC ON STACK
00EC E5	1620		PUSH	HL	
00ED 2A E1 07	1630		LD	HL,(LSAV)	*RESTORE HL
00F0 C9	1640		RET		*RETURN TO USER
00F1	1650	*			
00F1	1660	**			
00F1	1670	*			
00F1	1680	*CLDIS CLEARS THE DATA AND ADDRESS DISPLAYS			
00F1	1690	*SETS THE KEYBOARD BUFFER=0 AND CLEARS THE			
00F1	1700	*KEYBOARD FLAGS			
00F1	1710	*			
00F1 3E 00	1720	CLDIS	LD	A,ZERO	
00F3 32 F1 07	1730		LD	(KFLAGS),A	*CLEAR FLAGS
00F6 32 F2 07	1740		LD	(KDATA1),A	*CLEAR BUFFER
00F9 32 F3 07	1750		LD	(KDATA2),A	
00FC D3 07	1760		OUT	DATDIS	*CLEAR DATA FIELD DISPLAY
00FE D3 05	1770		OUT	ADDIS1	*CLEAR ADDRESS FIELD DISPLAY
0100 D3 06	1780		OUT	ADDIS2	
0102 C9	1790		RET		
0103	1800	*			
0103	1810	*			
0103	1820	*KEYIN WAITS FOR INPUT FROM THE KEYBOARD			
0103	1830	*UPON DETECTING DATA AT THE INPUT PORT (0)			
0103	1840	*VIA THE STROBE BIT (7) BEING SET THE DATA			
0103	1850	*IS INPUT,THE STROBE BIT CLEARED, AND THE INPUT			
0103	1860	*CHARACTER IS RETURNED TO THE USER IN A			
0103	1870	*			
0103	1880	*			
0103 DB 00	1890	KEYIN	IN	KEYPT	*INPUT DATA
0105 CB 7F	1900		BIT	7,A	
0107 CA 03 01	1910		JP	Z,KEYIN	*LOOP IF NO DATA
010A 32 F4 07	1911		LD	(TEMP),A	*SAVE CHARACTER

0100 DB 00	1912 KEYINI IN KEYPT	
010F CB 7F	1913 BIT 7,A	
0111 C2 0D 01	1914 JP NZ,KEYINI	*JUMP IF STROBE PRESENT
0114 3A F4 07	1915 LD A,(TEMP)	
0117 CB BF	1920 RES 7,A	*CLEAR STROBE
0119 C9	1930 RET	
011A *	1940 *	
011A *	1950 *	
011A *	1960 *KFLG02 SETS THE NEXT(0) AD NO DATA(2) KEYBOARD FLAGS	
011A *	1970 *	
011A *	1980 *	
011A 21 F1 07	1990 KFLG02 LD HL,KFLAGS	
011B CB CA	2000 SET 0,(HL)	*SET NEXT FLAG
011F CB D6	2010 SET 2,(HL)	
0121 E1	2020 POP HL	*CLEAR RETURN
0122 C9	2030 RET	
0123 *	2040 *	
0123 *	2050 *	
0123 *	2060 *KFLG0 SETS THE NEXT(0) KEYBOARD FLAG	
0123 *	2070 *	
0123 *	2080 *	
0123 21 F1 07	2090 KFLG0 LD HL,KFLAGS	
0126 CB CA	2100 SET 0,(HL)	*SET NEXT FLAG
0128 E1	2110 POP HL	*CLEAR RETURN
0129 C9	2120 RET	
012A *	2130 *	
012A *	2140 *	
012A *	2150 *KFLG12 SETS THE EXEC(1) AND NO DATA(2) KEYBOARD FLAG	
012A *	2160 *	
012A *	2170 *	
012A 21 F1 07	2180 KFLG12 LD HL,KFLAGS	
012D CB CE	2190 SET 1,(HL)	
012F CB D6	2200 SET 2,(HL)	
0131 E1	2210 POP HL	*CLEAR RETURN
0132 C9	2220 RET	
0133 *	2230 *	
0133 *	2240 *	
0133 *	2250 *KFLG1 SETS THE EXEC (1) KEYBOARD FLAG	
0133 *	2260 *	
0133 *	2270 *	
0133 21 F1 07	2280 KFLG1 LD HL,KFLAGS	
0136 CB CE	2290 SET 1,(HL)	*SET EXEC FLAG
0138 E1	2300 POP HL	*CLEAR RETURN
0139 C9	2310 RET	
013A *	2320 *	
013A *	2330 *	
013A *	2340 *	
013A *	2350 *ONECAR INPUTS ONE CHARACTER FOLLOWED BY A NEXT OR EXEC	
013A *	2360 *FROM THE KEYBOARD, VALIDATES IT, AND RETURNS IT TO	
013A *	2370 *THE USER IN KDATA2	
013A *	2380 *	
013A *	2390 *	
013A CD F1 00	2400 ONECAR CALL CLDIS	*CLEAR DISPLAY,BUFFER,&FLAGS
013D CB 03 01	2410 CALL KEYIN	*GET CHARACTER
0140 D3 07	2420 OUT DATDIS	*DISPLAY CHARACTER
0142 CB 5D 01	2430 CALL CARCK1	*CHECK CHARACTER
0145 CB 77	2440 BIT 6,A	
0147 C2 51 01	2450 JP NZ,ONECA1	*JUMP IF SHIFT
014A D6 10	2460 SUB 1,D	*CHARACTER=0-F
014C F2 3A 01	2470 JP P,ONECAR	*JUMP IF NOT 0-F
014F C6 10	2480 ADD 1,D	
0151 32 F3 07	2490 ONECA1 LD (KDATA2),A	*SAVE CHARACTER
0154 CD 03 01	2500 CALL KEYIN	*GET NEXT CHARACTER
0157 CD 6A 01	2510 CALL CARCK2	
015A C3 51 01	2520 JP ONECA1	*GO DO AGAIN NOT EXEC OR NEXT
015D *	2530 *	
015D *	2540 *	
015D *	2550 *CARCK1 CHECKS FOR A NEXT OR EXEC ON AN INITIAL	
015D *	2560 *CHARACTER. IF NEXT THE ROUTINE RETURNS TO CALLER VIA	

```

015D 2570 *KFLG02. IF EXEC THE ROUTINE RETURNS TO THE CALLER
015D 2580 *VIA KFLG12
015D 2590 *
015D 2600 *
015D 06 20 2610 CARCK1 LD B+NEXTC *CHECK FOR NEXT
015F 88 2620 CP B
0160 CA 1A 01 2630 JP Z+KFLG02 *IF NEXT JUMP
0163 06 10 2640 LD B+EXECC *CHECK FOR EXEC
0165 88 2650 CP B
0166 CA 2A 01 2660 JP Z+KFLG12 *IF EXEC JUMP
0169 C9 2670 RET *ELSE RETURN
016A 2680 *
016A 2690 *
016A 2700 *CARCK2 CHECKS FOR NEXT OR EXEC. SETS THE PROPER
016A 2710 *FLAG VIA *KFLG0 OR KFLG1 AND RETURNS TO THE USER
016A 2720 *IF NOT NEXT OR EXEC THE ROUTINE RETURNS TO
016A 2730 *THE ORIGINATOR OF THE REQUEST
016A 2740 *
016A 2750 *
016A 06 20 2760 CARCK2 LD B+NEXTC *CHECK FOR NEXT
016C 88 2770 CP B
016D CA 23 01 2780 JP Z+KFLG0 *IF NEXT JUMP
0170 06 10 2790 LD B+EXECC *CHECK FOR EXEC
0172 88 2800 CP B
0173 CA 33 01 2810 JP Z+KFLG1
017A C9 2820 RET
0177 2830 *
0177 2840 *
0177 2850 *TWOCAR INPUTS 2 CHARACTERS FROM THE KEYBOARD
0177 2860 *FOLLOWED BY A NEXT OR EXEC AND RETURNS THEM TO THE
0177 2870 *USER IN KDATA2
0177 2880 *
0177 2890 *
0177 CB 00 01 2900 TWOCAR CALL CLDAT *CLEAR BUFFER, FLAGS, AND DISPLAY
017A CD 03 01 2910 CALL KEYIN *GET CHARACTER
017D CD 50 01 2920 CALL CARCK1 *CHECK FOR NEXT OR EXEC
0180 06 10 2930 TWOCAR SUB IAD *CHARACTER-O-F
0182 F3 77 01 2940 JR F+TWOCAR *JUMP IF NOT O-F
0185 CA 10 2950 ADD IAD
0187 21 F3 07 2960 LD HL, KDATA2
018A 46 2970 LD B, (HL) *GET OLD DATA
018B CB 00 2980 RLC B
018D CB 00 2990 RLC B
018F CB 00 3000 RLC B
0191 CB 00 3010 RLC B
0193 80 3020 ADD A, B *A=OLD+NEW
0194 D3 07 3030 OUT DATDIS *DISPLAY INPUT
0196 77 3040 LD (HL), A *SAVE NEW DATA
0197 CD 03 01 3050 CALL KEYIN *GET NEXT CHARACTER
019A CD 6A 01 3060 CALL CARCK2 *CHECK FOR TERMINATION
019D C3 80 01 3070 JP TWOCAR *JUMP IF NO TERMINATION
01A0 3080 *
01A0 3090 *
01A0 3100 *CLDAT CLEARS THE INPUT BUFFER, FLAGS, AND DATA DIS
01A0 3110 *
01A0 3120 *
01A0 3E 00 3130 CLDAT LD A, ZERO
01A2 32 F1 07 3140 LD (KFLG0), A *CLEAR FLAGS
01A5 32 F3 07 3150 LD (KDATA2), A *CLEAR BUFFER
01A8 32 F2 07 3160 LD (KDATA1), A
01AB C9 3170 RET
01AC 3E 00 3180 CLADD LD A, ZERO *CLEAR ADDRESS DISPLAY
01AE D3 05 3182 OUT ADDIS1
01B0 D3 06 3183 OUT ADDIS2
01B2 C9 3184 RET
01B3 3190 *
01B3 3200 *
01B3 3210 *FORCAR INPUTS FOUR CHARACTERS FROM THE KEYBOARD
01B3 3220 *FOLLOWED BY A NEXT OR EXEC AND RETURNS THEM

```

01B3	3230	*TO THE USER IN KDATA1 AND KDATA2	
01B3	3240	*	
01B3	3250	*	
01B3 C0 A0 01	3260	FORCAR CALL CLDAT	*CLEAR FLAGS AND BUFFER
01B6 C0 03 01	3270	CALL KEYIN	*GET INPUT CHARACTER
01B9 C0 5D 01	3280	CALL CARCK1	*CHECK FOR NEXT OR EXEC
01BC D6 10	3290	FORCA1 SUB 16D	*CHARACTER=0-F
01BE F2 B3 01	3300	JP P+FORCAR	*JUMP IF NOT 0-F
01C1 C6 10	3310	ADD 16D	
01C3 32 F4 07	3320	LD (TEMP),A	*SAVE CHARACTER
01C6 3A F2 07	3330	LD A,(KDATA1)	*A=MSD
01C9 21 F3 07	3340	LD HL,KDATA2	
01CC E0 67	3350	RDD	*ADJUST DATA FOR NEW CHARACTER
01CE 07	3360	RLCA	
01CF 07	3370	RLCA	
01D0 07	3380	RLCA	
01D1 07	3390	RLCA	
01D2 E6 F0	3400	AND 240D	*MASK OFF OLD IDGIT
01D4 21 F4 07	3410	LD HL,TEMP	
01D7 86	3420	ADD A,(HL)	*ADD IN NEW DIGIT
01D8 2A F3 07	3430	LD HL,(KDATA2)	*SAVE NEW LSDS
01DB 22 F2 07	3440	LD (KDATA1),HL	*SAVE NEW MSDS
01DE 32 F3 07	3450	LD (KDATA2),A	*SAVE NEW LSDS
01E1 D3 06	3460	OUT ADDIS2	*DISPLAY LSDS
01E3 3A F2 07	3470	LD A,(KDATA1)	
01E6 D3 05	3480	OUT ADDIS1	
01E9 C0 03 01	3490	CALL KEYIN	*GET NEXT CHARACTER
01EB C0 6A 01	3500	CALL CARCK2	*CHECK FOR NEXT OR EXEC
01EE C3 BC 01	3510	JP FORCA1	*JUMP IF NOT NEXT OR EXEC
01F1	3520	*	
01F1	3530	*	
01F1	3540	*	
01F1	3550	*	
01F1	3560	*MEMORY INPUTS AN ADDRESS FROM THE KEYBOARD FOLLOWED	
01F1	3570	*BY DATA AS DEFINED BY THE SEQUENCE	
01F1	3580	* MEM(ADDRESS)NEXT,(DATA)NEXT,...(DATA)EXEC	
01F1	3590	*IF DATA IS TO BE DISPLAYED	
01F1	3600	* MEM(ADDRESS)NEXT,NEXT,...NEXT,EXEC	
01F1	3610	*EXEC WILL RETURN CONTROL TO THE COMMAND RECOGNITION	
01F1	3620	*	
01F1	3630	*	
01F1 3E 00	3640	MEMORY LD A,ZERO	*CLEAR MEMORY BASE ADDRESS
01F3 32 F6 07	3650	LD (MBASE1),A	
01F6 32 F7 07	3660	LD (MBASE2),A	
01F9 C0 AC 01	3661	CALL CLADD	
01FC C0 B3 01	3670	CALL FORCAR	*GET BASE ADDRESS
01FF 3A F1 07	3680	LD A,(KFLAGS)	
0202 C0 4F	3690	BIT 1,A	
0204 C2 B9 00	3700	JP NZ,MARK2	*JUMP IF EXEC FLAG SET
0207 3A F2 07	3710	LD A,(KDATA1)	*SAVE MEMORY ADDRESS
020A 32 F7 07	3720	LD (MBASE2),A	
020D 3A F3 07	3730	LD A,(KDATA2)	
0210 32 F6 07	3740	LD (MBASE1),A	
0213 2A F6 07	3750	LD HL,(MBASE1)	*GET MEM BASE ADDRESS
0216 7E	3760	HEM1 LD A,(HL)	*GET MEMORY DATA
0217 D3 07	3770	OUT DATD15	*DISPLAY MEMORY DATA
0219 C0 77 01	3780	CALL TWOCAR	*GET NEW DATA
021C 3A F1 07	3790	LD A,(KFLAGS)	
021F C0 57	3800	BIT 2,A	
0221 C2 43 02	3810	JP NZ,HEM2	*JUMP IF NO DATA
0224 2A F6 07	3820	LD HL,(MBASE1)	*GET MEM ADDRESS
0227 3A F3 07	3830	LD A,(KDATA2)	*GET NEW DATA
022A 77	3840	LD (HL),A	*REPLACE OLD DATA
022B 3A F1 07	3850	LD A,(KFLAGS)	
022E C0 4F	3860	BIT 1,A	
0230 C2 B9 00	3870	JP NZ,MARK2	*JUMP IF EXEC FLAG SET
0233 2A F6 07	3880	HEM12 LD HL,(MBASE1)	*INC BASE MEM ADD
0236 23	3890	INC HL	
0237 22 F6 07	3900	LD (MBASE1),HL	

023A 7D	3901	LD	A,L	
023B D3 06	3902	OUT	ADDIS2	
023D 7C	3903	LD	A,H	
023E D3 05	3904	OUT	ADDIS1	
0240 C3 16 02	3910	JP	MEM1	
0243 CB 4F	3920 MEM2	BIT	1,A	
0245 G2 89 00	3930	JP	NZ,WARM2	*JUMP IF EXEC FLAG SET
0248 C3 33 02	3940	JP	MEM12	
024B	3950 *			
024B	3960 *			
024B	3970 *			
024B	3980 *			
024B	3990	*REGISTER INPUTS A REGISTER FROM THE KEYBOARD FOLLOWED BY		
024B	4000	*DATA AS DEFINED BY THE SEQUENCE		
024B	4010	* REG(INIT REG)NEXT,(DATA)NEXT...(DATA)EXEC		
024B	4020	*REGISTER SEQUENCE IS IX,IY,SP,PC,I,R,H,L,A,B,C,D,E,F,		
024B	4030	*AL,AH,AA,AB,AC,AD,AE,AF		
024B	4040	*IF ONLY DATA IS TO BE DISPLAYED		
024B	4050	* REG(INIT REG)NEXT,NEXT...EXEC		
024B	4060	*EXEC WILL RETURN CONTROL TO THE COMMAND RECOGNITION		
024B	4070 *			
024B	4080 *			
024B CD 3A 01	4090	REGIST CALL	ONECAR	*GET INITIAL CHARACTER
024E 3A F1 07	4100	LD	A,(KFLAGS)	
0251 CB 57	4110	BIT	2,A	
0253 C2 89 00	4120	JP	NZ,WARM2	*JUMP IF NO DATA FLAG SET
0256 3A F3 07	4130	LD	A,(KDATA2)	*GET BASE REGISTER
0259 32 F5 07	4140	REGIO LD	(TEMP2),A	
025C CB 77	4141	BIT	5,A	*CHECK FOR SHIFT
025E C2 CC 02	4142	JP	NZ,REGISA	*JUMP IF SHIFT KEY SET
0261 FE 06	4143	CP	6	
0263 F2 6C 02	4144	JP	P,REGI1	*JUMP IF EIGHT BIT REGISTER
0266 3D	4145	DEC	A	
0267 3D	4146	DEC	A	
0268 87	4147	ADD	A	*I=(I-2)*2
0269 C3 6E 02	4148	JP	REGI2	
026C 3C	4149	REGI1 INC	A	
026D 3C	4150	INC	A	
026E 32 FB 07	4151	REGI2 LD	(REGINX),A	*SAVE INDEX
0271 3A F5 07	4152	LD	A,(TEMP2)	
0274 FE 10	4153	CP	10H	
0276 FA B3 02	4154	JP	M,REGI2A	
0279 CB 77	4155	BIT	6,A	
027B C2 83 02	4157	JP	NZ,REGI2A	*JUMP IF BIT 6 SET
027E 3E 48	4158	LD	A,40H	
0280 32 F5 07	4159	LD	(TEMP2),A	
0283 D3 07	4160	REGI2A OUT	DATDIS	*DISPLAY REGISTER SELECT
0285 3A F8 07	4210	LD	A,(REGINX)	
0288 FE 08	4220	CP	EIGHT	
028A FA D6 02	4230	JP	M,XYSP	*JUMP IF 16 BIT REG
028D 21 D7 07	4240	LD	HL,(XLSAV)	*GET BASE ADD
0290 4F	4250	LD	C,A	
0291 06 00	4260	LD	B,ZERO	
0293 09	4270	ADD	HL,BC	
0294 22 F6 07	4280	LD	(MBASE1),HL	*SAVE REG SAVE ADD
0297 7E	4290	LD	A,(HL)	*GET REGISTER DATA
0298 D3 06	4300	OUT	ADDIS2	*DISPLAY DATA
029A 78	4310	LD	A,B	
029B D3 05	4320	OUT	ADDIS1	
029D CB 77 01	4330	CALL	TWOCAR	*GET NEW DATA
02A0 3A F1 07	4340	LD	A,(NFLAGS)	
02A3 CB 57	4350	BIT	2,A	
02A5 C2 B7 02	4360	JP	NZ,REGI3	*JUMP IF NO DATA
02A8 2A F6 07	4390	LD	HL,(MBASE1)	
02AB 3A F2 07	4400	LD	A,(KDATA1)	*GET NEW DATA
02AE 77	4410	LD	(HL),A	*REPLACE OLD DATA
02AF 3A F1 07	4411	LD	A,(KFLAGS)	
02B2 CB 4F	4412	BIT	1,A	

02B4 C2 89 00	4413 JP NZ,WARM2	*JUMP IF EXEC FLAG SET
02B7 3A F5 07	4420 REGI3 LD A,(TEMP2)	*INCREMENT INDEX
02BA 3C	4421 INC A	
02BB 32 F5 07	4422 LD (TEMP2),A	
02BE 3A F8 07	4423 LD A,(REGINX)	*INCREMENT INDEX
02C1 3C	4430 INC A	
02C2 FE 1A	4440 CP 1AH	
02C4 FA 6E 02	4450 JP M,REGI2	*JUMP IF INDEX .LT. 1A
02C7 3E 02	4460 REGI4 LD A,TWO	*SET INITIAL INDEX
02C9 3A 59 02	4470 JP REGI0	
02CC 1A 48	4480 REGISA SUB 4BH	
02CE FA 4B 02	4490 JP M,REGIST	*JUMP IF INVALID REGISTER
02D1 C6 12	4500 ADD 12H	
02D3 C3 6E 02	4510 JP REGI2	
02D6 21 D7 07	4520 XYSP LD HL,(XLSAV	
02D9 4F	4530 LD C,A	
02DA 06 00	4540 LD B,ZERO	
02DC 09	4550 ADD HL,BC	*HL=REG SAVE ADDRESS
02DD 22 F6 07	4560 LD (MBASE1),HL	
02E0 7E	4570 LD A,(HL)	*DISPLAY REGISTER DATA
02E1 D3 06	4580 OUT ADDIS2	
02E3 23	4590 INC HL	
02E4 7E	4600 LD A,(HL)	
02E5 D3 05	4610 OUT ADDIS1	
02E7 3A FB 07	4620 LD A,(REGINX)	
02EA 3C	4630 INC A	
02EB 32 FB 07	4640 LD (REGINX),A	
02EE CD B3 01	4650 CALL FORCAR	*GET NEW DATA
02F1 3A F1 07	4660 LD A,(KFLAGS)	
02F4 CB 57	4670 BIT 2,A	
02FA C2 08 03	4680 JP NZ,REGIS	*JUMP IF NO DATA
02F9 2A FA 07	4710 LD HL,(MBASE1)	*REPLACE OLD DATA
02FC 3A F3 07	4720 LD A,(KDATA2)	
02FF 77	4730 LD (HL),A	
0300 3A F2 07	4740 LD A,(KDATA1)	
0303 23	4750 INC HL	
0304 77	4760 LD (HL),A	
0305 3A F1 07	4761 LD A,(KFLAGS)	
0308 CB 4F	4762 REGIS BIT 1,A	
030A C2 B7 00	4763 JP NZ,WARM2	*JUMP IF EXEC FLAG SET
030D C3 B7 02	4770 JP REGI3	
0310	4780 *	
0310	4790 *	
0310	4800 *	
0310	4810 *	
0310	4820 *GO RESETS THE USERS RESTART ADDRESS IN THE	
0310	4830 *REGISTER SAVE AREA AND EXITS TO THE RESTART	
0310	4840 *MODULE	
0310	4850 *	
0310	4860 *	
0310 CD AC 01	4870 GOREQ CALL CLADD	
0313 CD B3 01	4871 CALL FORCAR	*GET RESTART ADDRESS
0316 3A F1 07	4880 LD A,(KFLAGS)	
0319 CB 57	4890 BIT 2,A	
031B C2 89 00	4900 JP NZ,WARM2	*IF NO DATA EXIT
031E 3A F3 07	4910 LD A,(KDATA2)	*SAVE NEW ADDRESS
0321 32 DD 07	4920 LD (PCLSAV),A	
0324 3A F2 07	4930 LD A,(KDATA1)	
0327 32 DE 07	4940 LD (PCHSAV),A	
032A C3 AA 00	4950 JP RESTR1	
032D	4960 *	
032D	4970 *	
032D	4980 *	
032D	4990 *UATST IS A UART LOOP CHECK ROUTINE	
032D	5000 *IT UTILIZES A LOOP WITH THE OUTPUT	
032D	5010 *PORT PATCHED TO THE INPUT PORT	
032D	5020 *IF AN ERROR IS DETECTED THE ERROR IS	
032D	5030 *DISPLAYED ON THE ADDRESS DISPLAY AND	

032D		5040	*THE CHARACTER IS DISPLAYED ON THE DATA DISPLAY	
032D		5050	*THE OUTPUT CHARACTER IS DISPLAYED ON THE MSD	
032D		5060	*OF THE ADDRESS DISPLAY	
032D		5070	*	
032D 04 00		5080	UATST LD B,ZERO	*GET STATUS
032F DB 03		5090	IN UATST	
0331 CB 47		5100	BIT 0,A	
0333 CA 53 03		5110	JP Z,UAR1	*JUMP IF XMIT BUFFER NOT EMPTY
0336 78		5120	UATST0 LD A,B	*GET OUTPUT CHARACTER
0337 D3 05		5130	OUT ADDIS1	
0339 D3 02		5140	OUT UARTIO	
033B DB 03		5150	UATST1 IN UATST	
033D CB 4F		5160	BIT 1,A	
033F CA 3B 03		5170	JP Z,UATST1	*JUMP IF NO DATA AVAILABLE
0342 E6 1C		5180	AND 1CH	
0344 C2 53 03		5190	JP NZ,UAR1	*JUMP IF PARITY ERROR
0347 DB 02		5200	IN UARTIO	*GET INPUT CHARACTER
0349 D3 07		5250	OUT DATDIS	
034B B8		5260	CP B	
034C C2 5A 03		5270	JP NZ,UAR2	*JUMP IF INPUT,NE,OUTPUT
034F 04		5280	INC B	
0350 C3 3A 03		5290	JP UATST0	
0353 D3 06		5300	UAR1 OUT ADDIS2	*DISPLAY UART STATUS
0355 DB 02		5310	IN UARTIO	*GET INPUT DATA
0357 D3 07		5320	OUT DATDIS	
0359 76		5330	HALT	
035A 3E 0F		5340	UAR2 LD A,CFH	
035C ED 79		5350	OUT (ADDIS2),A	
035E 76		5360	HALT	
035F		5370	*	
035F		5380	*	
035F		5390	*TTYINPUT DRIVER	
035F		5400	*INPUTS DATA INTO THE SPECIFIED BUFFER	
035F		5410	*INPUT IS TERMINATED WHEN A CARRIAGE RETURN	
035F		5420	*IS DETECTED OR THE NUMBER OF SPECIFIED CHARACTERS	
035F		5430	*HAVE BEEN INPUTED FROM THE TRANSMITTING DEVICE	
035F		5440	*	
035F 2A F9 07		5450	TTYINP LD HL,(TTYISF)	*GET BUFFER ADDRESS
0362 3A FD 07		5460	LD A,(TTYIC)	*GET NUMBER OF CHARACTERS
0365 47		5470	LD B,A	
0366 DB 03		5480	TTYIN1 IN UATST	*GET UART STATUS
0368 CB 4F		5490	BIT 1,A	
036A CA 66 03		5500	JP Z,TTYIN1	*JUMP IF NO DATA
036B E6 1C		5510	AND 1CH	
036F C2 9B 03		5520	JP NZ,TTYERR	*JUMP IF PARITY ERROR
0372 DB 02		5570	IN UARTIO	*GET INPUT CHARACTER
0374 77		5580	LD (HL),A	*SAFE CHARACTER IN USERS BUF
0375 FE 0D		5590	CP A,ODH	
0376 CA 91 03		5600	JP Z,TTYIN2	*JUMP IF CARRIAGE RETURN
0379 3E 01		5610	LD A,ONE	*SET OUTPUT CHARACTER COUNT
037B 22 FB 07		5620	TTYIN3 LD (TTYOBF),HL	*SET OUTPUT BUFFER ADDRESS
037E 32 FE 07		5630	LD (TTYOC),A	
0381 78		5631	LD A,B	
0382 32 F4 07		5632	LD (TEMP),A	
0385 CD 9E 03		5640	CALL TTYOUT	*GO OUTPUT CHARACTER
038B 3A F4 07		5641	LD A,(TEMP)	
038B 47		5642	LD B,A	
038C 05		5650	DEC B	
038D CB		5660	RET Z	*RETURN IF ALL CHARACTERS IN
038E C3 66 03		5670	JP TTYIN1	
0391 21 9C 03		5680	TTYIN2 LD HL,LF	*GET LINE FEED ADDRESS
0394 3E 02		5690	LD A,TWO	
0396 06 01		5700	LD B,ONE	
0398 C3 7B 03		5710	JP TTYIN3	
039B C9		5720	TTYERR RET	*RETURN WITH ERROR CODE IN A
039C 0D 0A		5730	LF DB 0DH,0AH	*LINE FEED/CARRIAGE RETURN
039E		5740	*	
039E		5750	*TTY OUTPUT DRIVER	

```

039E      5760 *TTYOUT OUTPUTS DATA FROM THE SPECIFIED
039E      5770 *USERS BUFFER TO THE UART. THE NUMBER OF
039E      5780 *USER SPECIFIED CHARACTERS ARE OUTPUT
039E      5790 *AND CONTROL RETURNED TO THE USER
039E      5800 *
039E 2A FB 07 5810 TTYOUT LD HL,(TTYOBF) *GET BUFFER ADDRESS
03A1 3A FE 07 5820 LD A,(TTYOC) *GET NUMBER OF CHARACTERS
03A4 47 5830 LD B,A
03A5 0E 00 5840 TTYOU1 LD C,ZERO
03A7 11 00 00 5850 LD DE,ZERO
03AA DB 03 5860 TTYO1 IN UARTST *GET STATUS
03AC CB 47 5870 BIT 0,A
03AE CA BC 03 5880 JP Z,TTYOU2 *JUMP IF BUFFER NOT EMPTY
03B1 7E 5890 LD A,(HL) *GET CHARACTER
03B2 D3 02 5900 OUT UARTIO *OUTPUT CHARACTER
03B4 05 5910 DEC B
03B5 3E 00 5920 LD A,ZERO
03B7 CB 5930 RET Z *RETURN IF BUFFER EMPTY
03B8 23 5931 INC HL
03B9 C3 A5 03 5940 JP TTYOU1
03BC 13 5950 TTYOU2 INC DE *TRY AGAIN DELAY
03BD 7B 5960 LD A,E
03BE FE 00 5970 CP ZERO
03C0 C2 BC 03 5980 JP NZ,TTYOU2
03C3 7A 5990 LD A,D
03C4 FE 00 6000 CP ZERO
03C6 C2 BC 03 6010 JP NZ,TTYOU2
03C9 0C 6020 INC C
03CA FE 05 6030 CP FIVE
03CC C2 AA 03 6040 JP NZ,TTYO1 *JUMP IF <LT.5 TRYS
03CF 3E 01 6050 LD A,ONE *ELSE RETURN WITH A=1
03D1 C9 6060 RET
03D2 6070 *
07C4 6080 ST 7C4H
07C4 6090 *
07C4 6100 *PAGE 2 CONSTANTS,JUMP AREAS,AND REGISTER
07C4 6110 *SAVE AREAS
07C4 6120 *
07C4 6130 SPSTRT DB 0 *STACK AREA
00
07C5 6140 *
07C5 6150 * USER RESTART AREA
07C5 6160 *
07C5 6170 RST2V DB 3 *USER BRANCH AREA FOR RST 2
07C8 6180 RST3V DB 3 *USER BRANCH AREA FOR RST 3
07CB 6190 RST4V DB 3 *USER BRANCH AREA FOR RST 4
07CE 6200 RST5V DB 3 *USER BRANCH AREA FOR RST 5
07D1 6210 RST6V DB 3 *USER BRANCH AREA FOR RST 6
07D4 6220 RST7V DB 3 *USER BRANCH AREA FOR RST 7
07D7 6230 *
07D7 6240 *REGISTER SAVE AREA
07D7 6250 *
07D7 6260 IXLSAV DB 0
00
07DB 6270 IXHSV DB 0
00
07D7 6280 IYLSV DB 0
00
07DA 6290 IYHSV DB 0
00
07DB 6300 SPLSAV DB 0
00
07DC 6310 SPHSAV DB 0
00
07DD 6320 PCLSAV DB 0
00
07DE 6330 PCHSAV DB 0
00
07DF 6340 ISAV DB 0

```

00				
07E0	6350	RSBV	DB	0
00				
07E1	6360	LSBV	DB	0
00				
07E2	6370	HSBV	DB	0
00				
07E3	6380	ASBV	DB	0
00				
07E4	6390	BSBV	DB	0
00				
07E5	6400	CSBV	DB	0
00				
07E6	6410	DSBV	DB	0
00				
07E7	6420	ESBV	DB	0
00				
07E8	6430	FSBV	DB	0
00				
07E9	6440	ASBV	DB	0
00				
07EA	6450	AHSBV	DB	0
00				
07EB	6460	ABSBV	DB	0
00				
07EC	6470	ABSBV	DB	0
00				
07ED	6480	ACSBV	DB	0
00				
07EE	6490	ADSBV	DB	0
00				
07EF	6500	AESBV	DB	0
00				
07F0	6510	AFSBV	DB	0
00				
07F1	6520	*		
07F1	6530	*DATA STORAGE AREA		
07F1	6540	*		
07F1	6550	KFLAGS	DB	0
00				*KEYBOARD FLAGS
07F2	6560	KDATA1	DB	0
00				*KEYBOARD INPUT BUFFER
07F3	6570	KDATA2	DB	0
00				
07F4	6580	TEMP	DB	0
00				
07F5	6581	TEMP2	DB	0
00				
07F6	6590	MBASE1	DB	0
00				*BASE MEMORY ADDRESS
07F7	6600	MBASE2	DB	0
00				
07F8	6610	REGINX	DB	0
00				*REGISTER INDEX
07F9	6620	TTYIBF	DS	2
07F9	6630	TTYOBF	DS	2
07FD	6640	TTYIC	DB	0
00				*TTY INPUT CHARACTER COUNT
07FE	6650	TTYOC	DB	0
00				*TTY OUTPUT CHARACTER COUNT
07FF	6660	*		
07FF	6670	END		

FILE 3000 7323
READY

Appendix E Z80 CPU Technical Specifications

Appendix E1 Electrical Specifications

Absolute Maximum Ratings

Temperature Under Bias
Storage Temperature
Voltage On Any Pin
with Respect to Ground
Power Dissipation

Specified operating range:
-65°C to +150°C
-0.5V to +7V
1.5W

*Comments:

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and fractional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note: For Z80-CPU all AC and DC characteristics remain the same for the military grade parts except I_{CC} .

$I_{CC} = 200 \text{ mA}$

Z80-CPU D.C. Characteristics

$T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$ unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage		$V_{CC} - 0.5$	$V_{CC} + 3$	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	$I_{OL} = 1.8 \text{ mA}$
V_{OH}	Output High Voltage	2.4			V	$I_{OH} = -250 \mu\text{A}$
I_{CC}	Power Supply Current			150	mA	
I_{LI}	Input Leakage Current			10	μA	$V_{IN} = 0$ to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	$V_{OUT} = 2.4$ to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	$V_{OUT} = 0.4\text{V}$
I_{LB}	Data Bus Leakage Current in Input Mode			±10	μA	$0 < V_{IN} < V_{CC}$

Capacitance

$T_A = 25^\circ\text{C}$, $f = 1 \text{ MHz}$,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_{Φ}	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80-CPU

Ordering Information

C - Ceramic
P - Plastic
S - Standard 5V $\pm 5\%$ at 70°C
E - Extended 5V $\pm 5\%$ -40° to 85°C
M - Military 5V $\pm 10\%$ -55° to 125°C

Z80A-CPU D.C. Characteristics

$T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5\text{V} \pm 5\%$ unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage		$V_{CC} - 0.5$	$V_{CC} + 3$	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	$I_{OL} = 1.8 \text{ mA}$
V_{OH}	Output High Voltage	2.4			V	$I_{OH} = -250 \mu\text{A}$
I_{CC}	Power Supply Current		90	200	mA	
I_{LI}	Input Leakage Current			10	μA	$V_{IN} = 0$ to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	$V_{OUT} = 2.4$ to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	$V_{OUT} = 0.4\text{V}$
I_{LB}	Data Bus Leakage Current in Input Mode			±10	μA	$0 < V_{IN} < V_{CC}$

Capacitance

$T_A = 25^\circ\text{C}$, $f = 1 \text{ MHz}$,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_{Φ}	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80A-CPU

Ordering Information

C - Ceramic
P - Plastic
S - Standard 5V $\pm 5\%$ at 70°C

$T_A = 0^\circ\text{C to } 70^\circ\text{C}, V_{CC} = +5V \pm 5\%$, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
f	t_c	Clock Period	4	1.125	μs	
	$t_{w(HIGH)}$	Clock Pulse Width, Clock High	1.60	20	ns	
	$t_{w(LOW)}$	Clock Pulse Width, Clock Low	1.60	2000	ns	
	$t_{f, r}$	Clock Rise and Fall Time		70	ns	
A ₀₋₁₅	$t_{D(AD)}$	Address Output Delay		14.7	ns	
	$t_{F(AD)}$	Delay to Float		1.10	ns	
	t_{AC}	Address Setup Time to $\overline{RD}/\overline{WR}$ (Memory Cycle)	2.15		ns	$C_L = 50\text{pF}$
	t_{AS}	Address Setup Time to \overline{RD} , \overline{WR} , \overline{RD} or \overline{WR} (Data Cycle)	2.15		ns	
	t_{AH}	Address Setup Time to \overline{RD} or \overline{WR} (Timing Point)	5.65		ns	
D ₀₋₇	$t_{D(D)}$	Data Output Delay		220	ns	
	$t_{F(D)}$	Delay to Float During Write Cycle		30	ns	
	$t_{DS(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		ns	$C_L = 50\text{pF}$
	$t_{DS(D)}$	Data Setup Time to Falling Edge of Clock During M2 or M3	100		ns	
	t_{AC}	Data Setup Time to \overline{RD} (Memory Cycle)	2.15		ns	
	t_{AS}	Data Setup Time to \overline{RD} (Data Cycle)	10.5		ns	
	t_{AH}	Data Setup Time to \overline{RD}	2.15		ns	
H_L		Any Hold Time for Setup Time	0		ns	
\overline{RD}	$t_{D(\overline{RD})}$	\overline{RD} Delay From Rising Edge of Clock, \overline{RD} Low		100	ns	
	$t_{F(\overline{RD})}$	\overline{RD} Delay From Falling Edge of Clock, \overline{RD} High		120	ns	
	$t_{D(\overline{RD})}$	\overline{RD} Delay From Rising Edge of Clock, \overline{RD} High		100	ns	
	$t_{F(\overline{RD})}$	\overline{RD} Delay From Falling Edge of Clock, \overline{RD} Low		120	ns	
	$t_{w(\overline{RD})}$	Pulse Width, \overline{RD} Low	100		ns	
\overline{WR}	$t_{D(\overline{WR})}$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} Low		100	ns	
	$t_{F(\overline{WR})}$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} Low		120	ns	
	$t_{D(\overline{WR})}$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} High		100	ns	
	$t_{F(\overline{WR})}$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} High		120	ns	
	$t_{w(\overline{WR})}$	Pulse Width, \overline{WR} Low	100		ns	
\overline{RD}	$t_{D(\overline{RD})}$	\overline{RD} Delay From Rising Edge of Clock, \overline{RD} Low		100	ns	
	$t_{F(\overline{RD})}$	\overline{RD} Delay From Falling Edge of Clock, \overline{RD} Low		120	ns	
	$t_{D(\overline{RD})}$	\overline{RD} Delay From Rising Edge of Clock, \overline{RD} High		100	ns	
	$t_{F(\overline{RD})}$	\overline{RD} Delay From Falling Edge of Clock, \overline{RD} High		120	ns	
	$t_{w(\overline{RD})}$	Pulse Width, \overline{RD} Low	100		ns	
\overline{WR}	$t_{D(\overline{WR})}$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} Low		100	ns	
	$t_{F(\overline{WR})}$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} Low		120	ns	
	$t_{D(\overline{WR})}$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} High		100	ns	
	$t_{F(\overline{WR})}$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} High		120	ns	
	$t_{w(\overline{WR})}$	Pulse Width, \overline{WR} Low	100		ns	
M1	$t_{D(M1)}$	M1 Delay From Rising Edge of Clock, M1 Low		1.00	ns	$C_L = 50\text{pF}$
	$t_{F(M1)}$	M1 Delay From Rising Edge of Clock, M1 High		1.20	ns	
RFSH	$t_{D(RFSH)}$	RFSH Delay From Rising Edge of Clock, RFSH Low		100	ns	$C_L = 50\text{pF}$
	$t_{F(RFSH)}$	RFSH Delay From Rising Edge of Clock, RFSH High		120	ns	
WAIT	$t_s(WT)$	WAIT Setup Time to Rising Edge of Clock	70		ns	
HALT	$t_D(HT)$	HALT Delay Time From Falling Edge of Clock		300	ns	$C_L = 50\text{pF}$
INT	$t_s(INT)$	INT Setup Time to Rising Edge of Clock	60		ns	
NMI	$t_w(NMI)$	Pulse Width, NMI Low	80		ns	
BUSRD	$t_s(BR)$	BUSRD Setup Time to Rising Edge of Clock	60		ns	
BUSAK	$t_{D(BA)}$	BUSAK Delay From Rising Edge of Clock, BUSAK Low		120	ns	$C_L = 50\text{pF}$
	$t_{F(BA)}$	BUSAK Delay From Falling Edge of Clock, BUSAK High		120	ns	
RFSH	$t_s(RS)$	RFSH Setup Time to Rising Edge of Clock	60		ns	
\overline{F} (C)		Delay to Float (\overline{RD} , \overline{WR} , \overline{RD} and \overline{WR})		200	ns	
t_{MS}		M1 Stable Pulse to \overline{RD} (Memory Ack.)	1.15		ns	

NOTES:

A. Data should be enabled into the CPU data bus when \overline{RD} is active. During interrupt acknowledge data should be enabled when $\overline{M1}$ and \overline{RD} are both active.

B. All control signals are internally synchronized, as they may be totally asynchronous with respect to the clock.

C. The RESET signal must be active for a minimum of 1 clock cycle.

D. Output Delay vs. Load Capacitance

$T_A = 70^\circ\text{C}$

$V_{CC} = +5V \pm 5\%$

Add 10ns delay for each 50pF increase in load up to a maximum of 200pF for the data bus & 100pF for address & control lines.

E. Although static by design, timing parameters $t_{w(OUT)}$ or $t_{w(OUT)}$ of 700ns maximum.

$$(12) t_c = t_{w(OUT)} + t_{w(OUT)} + t_c + t_c$$

$$(11) t_{w(OUT)} + t_{w(OUT)} + t_c + t_c$$

$$(2) t_{w(OUT)} + t_c + t_c$$

$$(3) t_{w(OUT)} + t_c + t_c$$

$$(4) t_{w(OUT)} + t_c + t_c$$

$$(5) t_{w(OUT)} + t_c + t_c$$

$$(6) t_{w(OUT)} + t_c + t_c$$

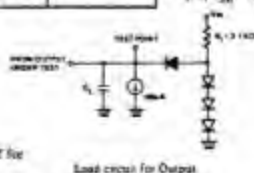
$$(7) t_{w(OUT)} + t_c + t_c$$

$$(8) t_{w(OUT)} + t_c + t_c$$

$$(9) t_{w(OUT)} + t_c + t_c$$

$$(10) t_{w(OUT)} + t_c + t_c$$

$$(11) t_{w(OUT)} + t_c + t_c$$



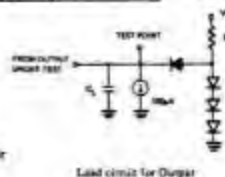
Load circuit for Output

$T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = +5V \pm 5\%$, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
+	t_c	Clock Period	25	33.3	μs	
	$t_{WH}(\Phi H)$	Clock Pulse Width, Clock High	150	15.1	μs	
	$t_{WL}(\Phi L)$	Clock Pulse Width, Clock Low	150	1000	μs	
	$t_{r,f}$	Clock Rise and Fall Time		30	μs	
	t_{AOZ}	Address Output Delay Delay to Float Address Stable Prior to \overline{RD} (Memory) Address Stable Prior to \overline{RD} , \overline{WR} , \overline{RD} (I/O Cycle) Address Stable from \overline{RD} , \overline{WR} , \overline{RD} or \overline{RD} Address Stable From \overline{RD} or \overline{WR} During Phase		150 75 150 150 150 150	μs	$C_L = 50\text{pF}$
\overline{RD}	$t_{DO}(\overline{RD})$	Data Output Delay		150	μs	
	$t_P(\overline{RD})$	Delay to First Data Write Cycle		50	μs	
	$t_{SQ}(\overline{RD})$	Data Setup Time to Rising Edge of Clock During \overline{RD} Cycle	30	50	μs	
	$t_{SD}(\overline{RD})$	Data Setup Time to Falling Edge of Clock During \overline{RD} Cycle	30	50	μs	
	t_{ACM}	Data Stable Prior to \overline{RD} (Memory Cycle)	150	150	μs	
	t_{AS}	Data Stable Prior to \overline{RD} (I/O Cycle)	150	150	μs	
	t_{DF}	Data Stable From \overline{RD}	150	150	μs	
\overline{WR}	t_H	Any Hold Time for Setup Time		0	μs	
	$t_{DLE}(\overline{WR})$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} Low		65	μs	
	$t_{DHF}(\overline{WR})$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} High		65	μs	
	$t_{DHF}(\overline{WR})$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} High		65	μs	
	t_{WHL}	Pulse Width, \overline{WR} Low	180	150	μs	
\overline{RD}	$t_{DLE}(\overline{RD})$	\overline{RD} Delay From Rising Edge of Clock, \overline{RD} Low		75	μs	
	$t_{DHF}(\overline{RD})$	\overline{RD} Delay From Falling Edge of Clock, \overline{RD} Low		65	μs	
	$t_{DHF}(\overline{RD})$	\overline{RD} Delay From Rising Edge of Clock, \overline{RD} High		65	μs	
	$t_{DHF}(\overline{RD})$	\overline{RD} Delay From Falling Edge of Clock, \overline{RD} High		65	μs	
	t_{RHL}	Pulse Width, \overline{RD} Low	180	150	μs	
\overline{WR}	$t_{DLE}(\overline{WR})$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} Low		65	μs	
	$t_{DHF}(\overline{WR})$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} Low		65	μs	
	$t_{DHF}(\overline{WR})$	\overline{WR} Delay From Rising Edge of Clock, \overline{WR} High		65	μs	
	$t_{DHF}(\overline{WR})$	\overline{WR} Delay From Falling Edge of Clock, \overline{WR} High		65	μs	
	t_{WHL}	Pulse Width, \overline{WR} Low	180	150	μs	
\overline{INT}	$t_{DI}(\overline{INT})$	\overline{INT} Delay From Rising Edge of Clock, \overline{INT} Low		100	μs	
	$t_{DI}(\overline{INT})$	\overline{INT} Delay From Rising Edge of Clock, \overline{INT} High		100	μs	
\overline{RFSH}	$t_{DI}(\overline{RFSH})$	\overline{RFSH} Delay From Rising Edge of Clock, \overline{RFSH} Low		150	μs	
	$t_{DI}(\overline{RFSH})$	\overline{RFSH} Delay From Rising Edge of Clock, \overline{RFSH} High		120	μs	
\overline{WAIT}	$t_s(\overline{WAIT})$	\overline{WAIT} Setup Time to Falling Edge of Clock	70		μs	
\overline{HALT}	$t_D(\overline{HALT})$	\overline{HALT} Delay Time From Falling Edge of Clock		300	μs	
\overline{INT}	$t_s(\overline{INT})$	\overline{INT} Setup Time to Rising Edge of Clock	80		μs	
\overline{RMI}	$t_{WH}(\overline{RMI})$	Pulse Width, \overline{RMI} Low	80		μs	
\overline{BUSRD}	$t_s(\overline{BUSRD})$	\overline{BUSRD} Setup Time to Rising Edge of Clock	50		μs	
\overline{BUSAK}	$t_{DI}(\overline{BUSAK})$	\overline{BUSAK} Delay From Rising Edge of Clock, \overline{BUSAK} Low		100	μs	
	$t_{DI}(\overline{BUSAK})$	\overline{BUSAK} Delay From Falling Edge of Clock, \overline{BUSAK} High		100	μs	
\overline{RESET}	$t_s(\overline{RESET})$	\overline{RESET} Setup Time to Rising Edge of Clock	80		μs	
	$t_P(\overline{C})$	Delay to Float (\overline{RD} , \overline{WR} , \overline{RD} and \overline{WR})		80	μs	
	t_{MS}	MS Stable Prior to \overline{RD} (Interrupt Ack.)	111		μs	

NOTES:

- Data should be enabled onto the CPU data bus when \overline{RD} is active. During interrupt acknowledge data should be enabled when \overline{INT} and \overline{RD} are both active.
- All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.
- The \overline{RESET} signal must be active for a maximum of 3 clock cycles.
- Output Delay vs. Loaded Capacitance
 $T_A = 70^\circ\text{C}$, $V_{CC} = +5V \pm 5\%$
Add 10nsec delay for each 50pF increase in load up to maximum of 200pF for data bus and 100pF for address & control lines.
- Although static by design, timing parameters $t_{WH}(\overline{INT})$ of 200 μs maximum.



$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

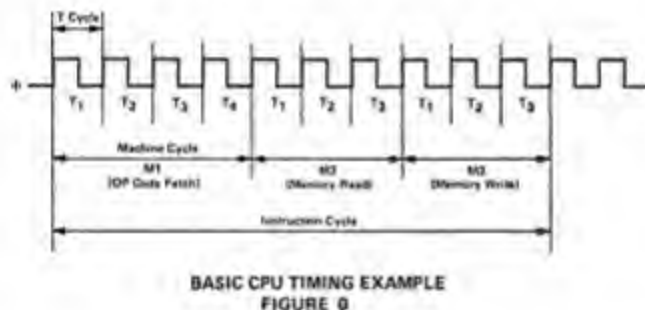
$$t_{WH}(\overline{INT}) = t_{WH}(\overline{INT}) + t_{WH}(\overline{INT})$$

Appendix E2 CPU Timing

The Z-80 CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

Memory read or write
I/O device read or write
Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T cycles and the basic operations are referred to as M (for machine) cycles. Figure 0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T cycles long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles.



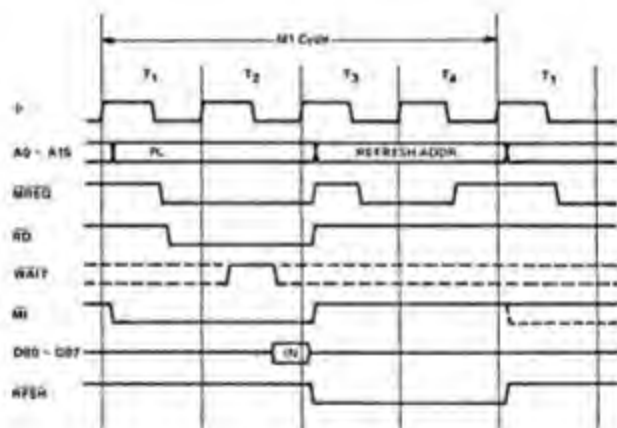
BASIC CPU TIMING EXAMPLE
FIGURE 0

All CPU timing can be broken down into a few very simple timing diagrams as shown in figure 1 through 7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices):

1. Instruction OP code fetch (M1 cycle)
2. Memory data read or write cycles
3. I/O read or write cycles
4. Bus Request/Acknowledge Cycle
5. Interrupt Request/Acknowledge Cycle
6. Non maskable Interrupt Request/Acknowledge Cycle
7. Exit from a HALT instruction

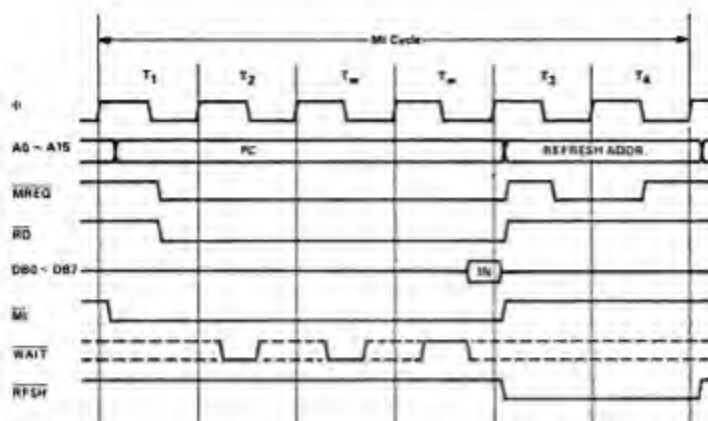
INSTRUCTION FETCH

Figure 1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the MREQ signal goes active. At this time the address to the memory has had time to stabilize so that the falling edge of MREQ can be used directly as a chip enable clock to dynamic memories. The RD line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the RD and MRQ signals. Thus the data has already been sampled by the CPU before the RD signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time). During T3 and T4 the lower 7 bits of the address bus contain a memory refresh address and the RFSH signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that a RD signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The MREQ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal can not be used by itself since the refresh address is only guaranteed to be stable during MREQ time.



INSTRUCTION OF CODE FETCH
FIGURE 1

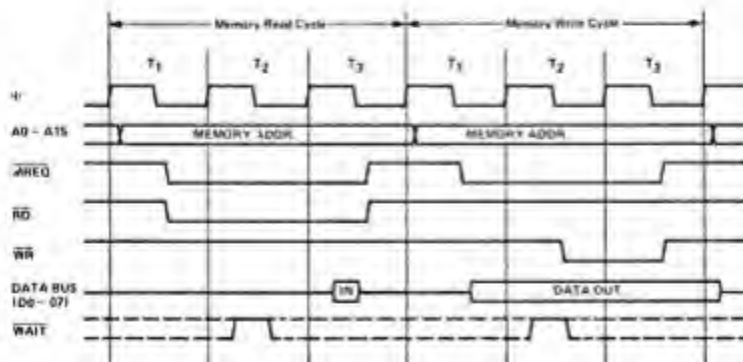
Figure 1A illustrates how the fetch cycle is delayed if the memory activates the WAIT line. During T2 and every subsequent Tw, the CPU samples the WAIT line with the falling edge of Φ . If the WAIT line is active at this time, another wait state will be entered during the following cycle. Using this technique the read cycle can be lengthened to match the access time of any type of memory device.



INSTRUCTION OF CODE FETCH WITH WAIT STATES
FIGURE 1A

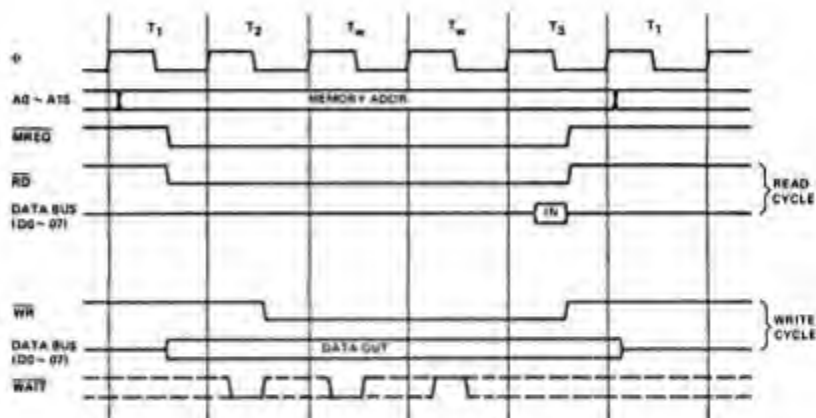
MEMORY READ OR WRITE

Figure 2 illustrates the timing of memory read or write cycles other than an OP code fetch (MI cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the WAIT signal. The MREQ signal and the RD signal are used the same as in the fetch cycle. In the case of a memory write cycle, the MREQ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The WR line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore the WR signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.



MEMORY READ OR WRITE CYCLES
FIGURE 2

Figure 2A illustrates how a WAIT request signal will lengthen any memory read or write operation. This operation is identical to that previously described for a fetch cycle. Notice in this figure that a separate read and a separate write cycle are shown in the same figure although read and write cycles can never occur simultaneously.



MEMORY READ OR WRITE CYCLES WITH WAIT STATES
FIGURE 2A

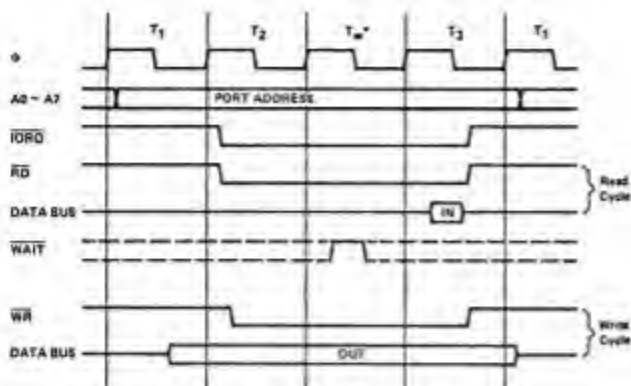
INPUT OR OUTPUT CYCLES

Figure 3 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason for this is that during I/O operations, the time from when the IORQ signal goes active until the CPU must sample the WAIT line is very short and without this extra state sufficient time does not exist for an I/O port to decode its address and activate the WAIT line if a wait is required. Also, without this wait state it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time the WAIT request signal is sampled. During a read I/O operation, the RD line is used to enable the addressed port onto the data bus just as in the case of a memory read. For I/O write operations, the WR line is used as a clock to the I/O port, again with sufficient overlap timing automatically provided so that the rising edge may be used as a data clock.

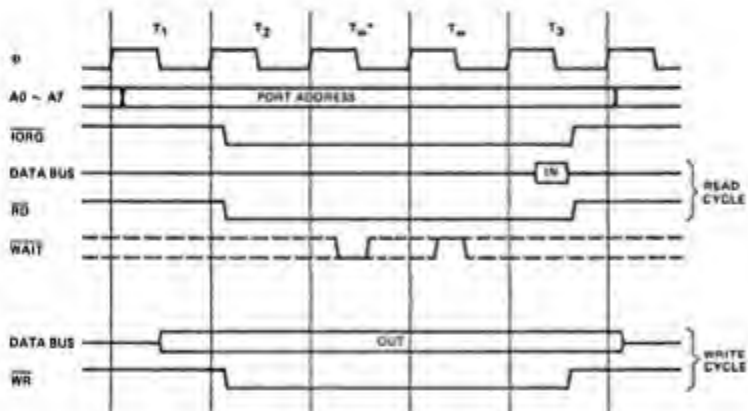
Figure 3A illustrates how additional wait states may be added with the WAIT line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

Figure 4 illustrates the timing for a Bus Request/Acknowledge cycle. The BUSRQ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the BUSRQ signal is active, the CPU will set its address, data and tri-state control signals to the high impedance state with the rising edge of the next clock pulse. At that time any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing). The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memories are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either a NMI or an INT signal.

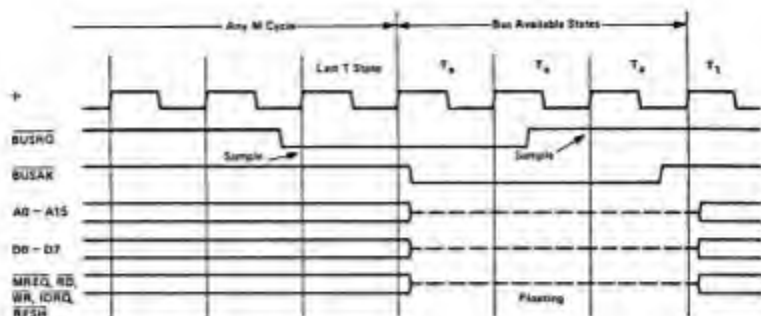


INPUT OR OUTPUT CYCLES
FIGURE 3



INPUT OR OUTPUT CYCLES WITH WAIT STATES
FIGURE 3A

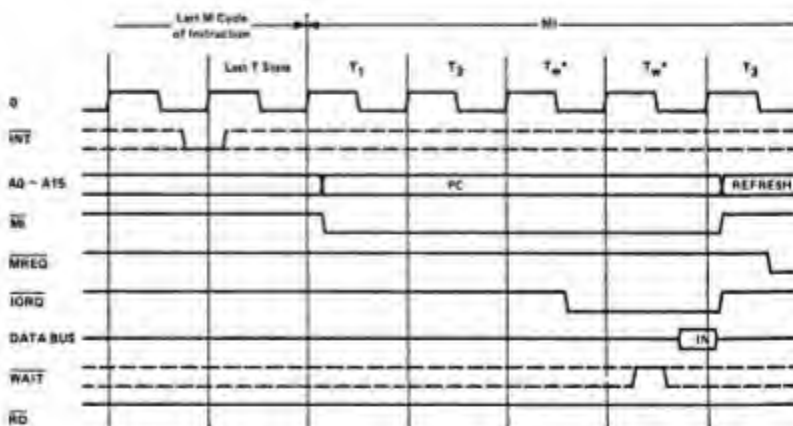
* Automatically inserted WAIT state



BUS REQUEST/ACKNOWLEDGE CYCLE
FIGURE 4

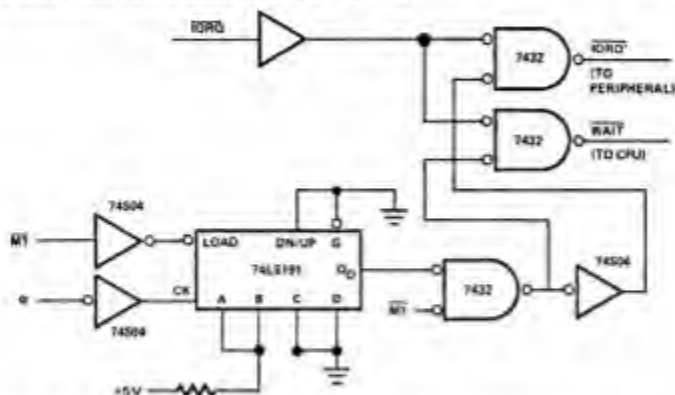
INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

Figure 5 illustrates the timing associated with an interrupt cycle. The interrupt signal (\overline{INT}) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the **BUSRQ** signal is active. When the signal is accepted a special M1 cycle is generated. During this special M1 cycle the **IORQ** signal becomes active (instead of the normal **MREQ**) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stabilize and identify which I/O device must insert the response vector.

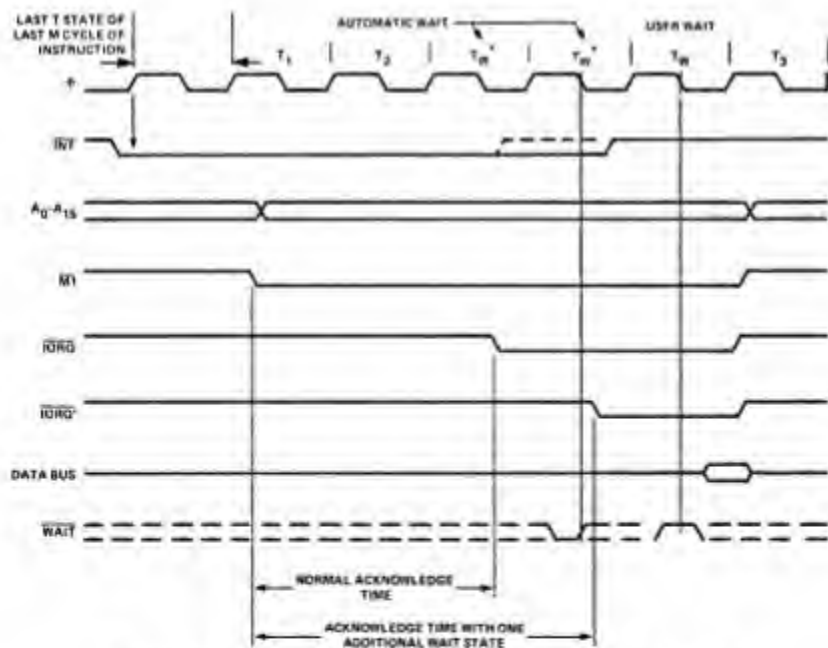


INTERRUPT REQUEST/ACKNOWLEDGE CYCLE
FIGURE 5

Figures 5A and 5B illustrate how a programmable counter can be used to extend interrupt acknowledge time. (Configured as shown to add one wait state)



EXTENDING INTERRUPT ACKNOWLEDGE TIME WITH WAIT STATE
FIGURE 5A



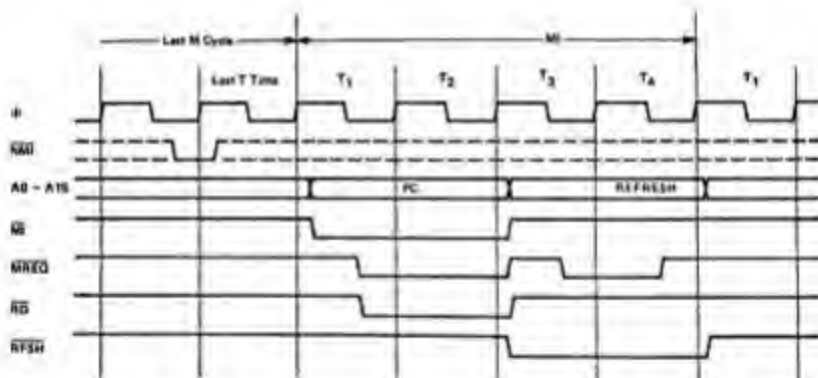
REQUEST/ACKNOWLEDGE CYCLE WITH ONE ADDITIONAL WAIT STATE
FIGURE 5B

NON MASKABLE INTERRUPT RESPONSE

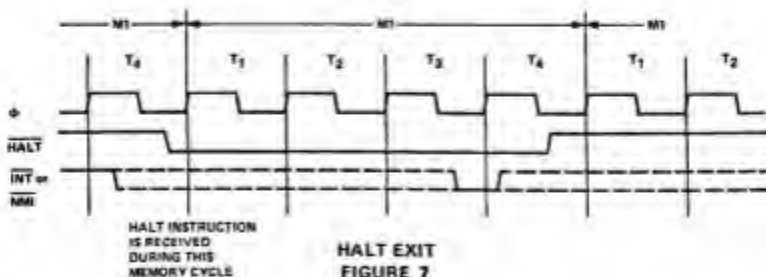
Figure 6 illustrates the request/acknowledge cycle for the non maskable interrupt. This signal is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non maskable interrupt is similar to a normal memory read operation. The only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066_H. The service routine for the non maskable interrupt must begin at this location if this interrupt is used.

HALT EXIT

Whenever a software halt instruction is executed the CPU begins executing NOP's until an interrupt is received (either a non maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state as shown in figure 7. If a non maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the halt state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non maskable one will be acknowledged since it has highest priority. The purpose of executing NOP instructions while in the halt state is to keep the memory refresh signals active. Each cycle in the halt state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and a NOP instruction is forced internally to the CPU. The halt acknowledge signal is active during this time to indicate that the processor is in the halt state.



NON MASKABLE INTERRUPT REQUEST OPERATION
FIGURE 6



HALT EXIT
FIGURE 7

Appendix E3 Instruction Set Summary



ADC HL, ss	Add with Carry Reg. pair ss to HL	CPL	Complement Acc. (1's comp)
ADC A, s	Add with carry operand s to Acc.	DAA	Decimal adjust Acc.
ADD A, n	Add value n to Acc.	DEC m	Decrement operand m
ADD A, r	Add Reg. r to Acc.	DEC IX	Decrement IX
ADD A, (HL)	Add location (HL) to Acc.	DEC IY	Decrement IY
ADD A, (IX+d)	Add location (IX+d) to Acc.	DEC ss	Decrement Reg. pair ss
ADD A, (IY+d)	Add location (IY+d) to Acc.	DI	Disable interrupts
ADD HL, ss	Add Reg. pair ss to HL	DJNZ e	Decrement B and Jump relative if B≠0
ADD IX, pp	Add Reg. pair pp to IX	EI	Enable interrupts
ADD IY, rr	Add Reg. pair rr to IY	EX (SP), HL	Exchange the location (SP) and HL
AND s	Logical 'AND' of operand s and Acc.	EX (SP), IX	Exchange the location (SP) and IX
BIT b, (HL)	Test BIT b of location (HL)	EX (SP), IY	Exchange the location (SP) and IY
BIT b, (IX+d)	Test BIT b of location (IX+d)	EX AF, AF'	Exchange the contents of AF and AF'
BIT b, (IY+d)	Test BIT b of location (IY+d)	EX DE, HL	Exchange the contents of DE and HL
BIT b, r	Test BIT b of Reg. r	EXX	Exchange the contents of BC, DE, HL with contents of BC', DE', HL' respectively
CALL cc, nn	Call subroutine at location nn if condition cc is true	HALT	HALT (wait for interrupt or reset)
CALL nn	Unconditional call subroutine at location nn	IM 0	Set interrupt mode 0
CCF	Complement carry flag	IM 1	Set interrupt mode 1
CP s	Compare operand s with Acc.	IM 2	Set interrupt mode 2
CPD	Compare location (HL) and Acc. decrement HL and BC	IN A, (n)	Load the Acc. with input from device n
CPDR	Compare location (HL) and Acc. decrement HL and BC, repeat until BC=0	IN r, (C)	Load the Reg. r with input from device (C)
CPI	Compare location (HL) and Acc. increment HL and decrement BC	INC (HL)	Increment location (HL)
CPIR	Compare location (HL) and Acc. increment HL, decrement BC repeat until BC=0	INC IX	Increment IX

INC (IX+d)	Increment location (IX+d)	LD HL, (nn)	Load HL with location (nn)
INC IY	Increment IY	LD (HL), r	Load location (HL) with Reg. r
INC (IY+d)	Increment location (IY+d)	LD I, A	Load I with Acc.
INC r	Increment Reg. r	LD IX, nn	Load IX with value nn
INC ss	Increment Reg. pair ss	LD IX, (nn)	Load IX with location (nn)
IND	Load location (HL) with input from port (C), decrement HL and B	LD (IX+d), n	Load location (IX+d) with value n
INDR	Load location (HL) with input from port (C), decrement HL and decrement B, repeat until B=0	LD (IX+d), r	Load location (IX+d) with Reg. r
INI	Load location (HL) with input from port (C); and increment HL and decrement B	LD IY, nn	Load IY with value nn
INIR	Load location (HL) with input from port (C), increment HL and decrement B, repeat until B=0	LD IY, (nn)	Load IY with location (nn)
JP (HL)	Unconditional Jump to (HL)	LD (IY+d), n	Load location (IY+d) with value n
JP (IX)	Unconditional Jump to (IX)	LD (IY+d), r	Load location (IY+d) with Reg. r
JP (IY)	Unconditional Jump to (IY)	LD (nn), A	Load location (nn) with Acc.
JP cc, nn	Jump to location nn if condition cc is true	LD (nn), dd	Load location (nn) with Reg. pair dd
JP nn	Unconditional jump to location nn	LD (nn), HL	Load location (nn) with HL
JP C, e	Jump relative to PC+e if carry=1	LD (nn), IX	Load location (nn) with IX
JR e	Unconditional Jump relative to PC+e	LD (nn), IY	Load location (nn) with IY
JP NC, e	Jump relative to PC+e if carry=0	LD R, A	Load R with Acc.
JR NZ, e	Jump relative to PC+e if non zero (Z=0)	LD r, (HL)	Load Reg. r with location (HL)
JR Z, e	Jump relative to PC+e if zero (Z=1)	LD r, (IX+d)	Load Reg. r with location (IX+d)
LD A, (BC)	Load Acc. with location (BC)	LD r, (IY+d)	Load Reg. r with location (IY+d)
LD A, (DE)	Load Acc. with location (DE)	LD r, n	Load Reg. r with value n
LD A, I	Load Acc. with I	LD r, r'	Load Reg. r with Reg. r'
LD A, (nn)	Load Acc. with location nn	LD SP, HL	Load SP with HL
LD A, R	Load Acc. with Reg. R	LD SP, IX	Load SP with IX
LD (BC), A	Load location (BC) with Acc.	LD SP, IY	Load SP with IY
LD (DE), A	Load location (DE) with Acc.	LDD	Load location (DE) with location (HL), decrement DE, HL and BC
LD (HL), n	Load location (HL) with value n	LDDR	Load location (DE) with location (HL), decrement DE, HL and BC; repeat until BC=0
LD dd, nn	Load Reg. pair dd with value nn	LDI	Load location (DE) with location (HL), increment DE, HL, decrement BC
		LDIR	Load location (DE) with location (HL), increment DE, HL, decrement BC and repeat until BC=0
		NEG	Negate Acc. (2's complement)
		NOP	No operation

OR s	Logical 'OR' or operand s and Acc.	RST p	Restart to location p
OTDR	Load output port (C) with location (HL), decrement HL and B, repeat until B=0	SBC A, s	Subtract operand s from Acc. with carry
OTIR	Load output port (C) with location (HL), increment HL, decrement B, repeat until B=0	SBC HL, ss	Subtract Reg. pair ss from HL with carry
OUT (C), r	Load output port (C) with Reg. r	SCF	Set carry flag (C=1)
OUT (n), A	Load output port (n) with Acc.	SET b, (HL)	Set Bit b of location (HL)
OUTD	Load output port (C) with location (HL), decrement HL and B	SET b, (IX+d)	Set Bit b of location (IX+d)
OUTI	Load output port (C) with location (HL), increment HL and decrement B	SET b, (IY+d)	Set Bit b of location (IY+d)
POP IX	Load IX with top of stack	SET b, r	Set Bit b of Reg. r
POP IY	Load IY with top of stack	SLA m	Shift operand m left arithmetic
POP qq	Load Reg. pair qq with top of stack	SRA m	Shift operand m right arithmetic
PUSH IX	Load IX onto stack	SRL m	Shift operand m right logical
PUSH IY	Load IY onto stack	SUB s	Subtract operand s from Acc.
PUSH qq	Load Reg. pair qq onto stack	XOR s	Exclusive 'OR' operand s and Acc.
RES b, m	Reset Bit b of operand m		
RET	Return from subroutine		
RET cc	Return from subroutine if condition cc is true		
RETI	Return from interrupt		
RETN	Return from non maskable interrupt		
RL m	Rotate left through carry operand m		
RLA	Rotate left Acc. through carry		
RLC (HL)	Rotate location (HL) left circular		
RLC (IX+d)	Rotate location (IX+d) left circular		
RLC (IY+d)	Rotate location (IY+d) left circular		
RLC r	Rotate Reg. r left circular		
RLCA	Rotate left circular Acc.		
RLD	Rotate digit left and right between Acc. and location (HL)		
RR m	Rotate right through carry operand m		
RRA	Rotate right Acc. through carry		
RRC m	Rotate operand m right circular		
RRCA	Rotate right circular Acc.		
RRD	Rotate digit right and left between Acc. and location (HL)		

GLOSSARY

Accumulator A temporary register where results of calculations may be stored by the central processor. One or more accumulators may be part of the arithmetic-logical unit.

Acoustical coupler A device that permits a terminal to be connected to the computer via a telephone line. It connects to the telephone handset.

Address An identifying number or label for locations in the memory.

Algorithm A step-by-step solution to a problem in a finite number of steps. A specific procedure for accomplishing a desired result.

ASCII American Standard Code for Information Interchange. Widely used 7-bit standard code. Also known as USASCII; IBM uses EBCDIC, which has 8 bits.

Assembler A program that converts symbolic instructions into machine macro-instructions.

Backplane A board equipped with plugs interconnected by buses into which the modules that make up a computer may be inserted. Also known as a motherboard.

BASIC Beginner's All-purpose Symbolic Instruction Code. Algebraic language developed at Dartmouth College. The language is easy to learn and use.

Binary A numbering system based on multiples of two using the digits 0 and 1.

Bit Abbreviation of binary digit. A single element in a binary number—either a 0 or a 1. Bits are represented in a microcomputer by the status of electronic switches that can be either on or off. Four bits equal a nibble; eight bits equal a byte.

Byte A group of adjacent bits, usually eight bits, which is operated upon as a unit by the central processor.

CMOS Complementary Metal-Oxide Semiconductor. Technology that combines the component density of p-channel MOS (PMOS) and the speed of n-channel MOS (NMOS). Power consumption is very low.

Clock A device that generates regular pulses that synchronize events throughout a microcomputer.

Central processor The central processor controls the operation of a microcomputer. The central processor can fetch and store data and instructions from memory.

CRT Cathode-Ray Tube. An electronic vacuum tube that can be used for graphic display. Also refers to a terminal incorporating a CRT.

Compiler A program that translates high-level programming language into machine language. May produce numerous macro-instructions for each high-level instruction, unlike an assembler which translates item for item. When using a compiler, one cannot change a program without recompilation.

Development system A microcomputer system having all the related equipment necessary for hardware and software development.

Digital Pertaining to discrete integral numbers in a given base which may express all

the variables occurring in a problem. Represented electronically by 2 (binary) to 16 (hexadecimal) states at the present time. Contrasts with analog, which refers to a continuous range of voltage or current quantities.

Double density Method of doubling bit density on magnetic storage mediums.

Dynamic memory Storage of data on dynamic chips in which storage of a small charge indicates a bit. Because the charge leaks over time, dynamic memory must be periodically refreshed.

EBCDIC IBM's 8-bit code, similar to ASCII.

Editor A program that rearranges text. Permits the addition or deletion of symbols and changes of format.

EIA-RS-232C Interface standard for data transmitted sequentially that is not synchronous with the central processor.

EPROM Erasable-Programmable Read-Only Memory. A PROM that can be erased and reprogrammed. Some EPROMs have a quartz window over the chip; data can be erased by exposure to intense ultraviolet light; other EPROMs may be erased electrically.

File A set of related records treated as a unit.

Flag A bit attached to a word for identification or for the purpose of signaling some condition. Typical microprocessors include carry, zero, sign, overflow and half-carry status flags.

Floating-point package A set of software routines that allows some microcomputers to perform floating-point arithmetic without the addition of extra hardware.

FSK Frequency Shift Keying. Technique of transforming bits into two different frequencies representing 0 and 1 for transmission over telephone or radio lines. The interface device is called a modem.

Ground Electrical reference point of a circuit.

Hard-copy Printed output on paper.

Hardware The physical components, peripherals, or other equipment that make up a computer system. Contrast with software.

Hexadecimal A numbering system based on multiples of 16 using the character 0 thru 9 and A thru F. For example, 0B hexadecimal equals 0000 1011 binary. One byte may be encoded in exactly 2 hexadecimal symbols.

High-level language A programming language that is relatively independent of assembler or machine language. The grammar often resembles English and requires a compiler or interpreter to convert to executable code. Examples: BASIC, FORTRAN, COBOL, ALGOL, PL/M, APL.

Instruction A step in a program that defines an operation together with the address(es) of any data needed for the operation.

Interface A common boundary between two systems or devices. The hardware or software necessary to interconnect two parts of a system.

Interrupt A break in the execution of a program usually caused by a signal from an

external device.

Kansas City standard Refers to a standard for cassette tape recordings of EIA-RS-232C data. Eight cycles of 2400 Hz equals 1, and 4 cycles of 1200 Hz equals 0.

Least significant bit The binary digit occupying the right-most position in a number or word, ie: 2^0 or 1.

LIFO Last-In, First-Out. Method of accessing the most recent entry, then the next most recent, and so on.

Light pen Photosensitive device that can be used to change the display on a CRT by generating a pulse at the point of contact.

Machine language Sets of binary integers that may be directly executed as instructions by the microcomputers without prior interpretation.

Mass storage Floppy disks, cassettes or tapes used to store large amounts of data. Less accessible, but larger than main storage.

Memory Storage device for binary information.

Microcomputer A small computer system capable of performing a basic repertoire of instructions. Includes a central processor, often contained on a single chip, memory, I/O devices, and power supply.

Microprocessor A central processor on a chip. A complete processor on a single chip, manufactured using microminiature manufacturing techniques, known as LSI (large scale integration).

Modem MODulator—DEModulator. Device that transforms binary data into frequencies suitable for transmission over telephone lines and back again.

Monitor A program that controls the operation of basic routines to optimize computer time.

Most significant bit The binary digit occupying the left-most position in a number or word, usually 2^7 or 128.

Octal A numbering system based on multiples of eight using digits 0 thru 7. Now largely superseded by the hexadecimal system.

Operating system Software that operates the hardware resources of a microcomputer. The operating system may do scheduling, debugging, I/O control, accounting, compilation, storage assignment, and data management.

Parity An extra bit that indicates whether a computer word has an odd or even number of 1s. Used to detect errors.

Peripheral Any piece of equipment, usually an I/O device, attached to the central processor.

Programmable memory Storage in which access to new information is independent of the address previously examined.

Read-only memory (ROM) Storage that cannot be altered. The information is written at the time of manufacture.

Register A memory device directly accessible by the central processor used for the

temporary storage of a computer word during arithmetic, logical, or input/output operations.

S-100 A 100-pin bus used in the popular 8080/Z80 system.

Software Programs that translate high-level languages into machine language, such as compilers, operating systems, assemblers, generators, library routines, and editors.

Stack A technique of presenting programs sequentially. A stack is a LIFO structure controlled by PUSH and POP instructions.

Tiny BASIC The BASIC programming language reduced to a simple form that permits integer arithmetic and some string operations. Tiny BASIC usually occupies 4 K or less bytes of memory.

Three-state Capable of existing in three logical states—0 (low), 1 (high), or undefined (high-impedance), ie: floating.

UART Universal Asynchronous Receiver Transmitter. A transmitter that converts serial to parallel and *vice versa*.

Word A set of bits that occupies one storage location and is treated as a unit. May have any number of bits, but usually 4, 8, or 16.

Word processor A text editor that allows the user to modify text: formats, books, letters, and reports.

INDEX

- Accumulators, [27, 33](#)
- ADC, [51, 63](#)
- ADD, [49, 63](#)
- Addressing, [29, 32-33, 98, 105](#)
 - capability, [32](#)
 - high-order, [32](#)
 - low-order, [32](#)
- AND, [34, 54](#)
- Arithmetic and Logic Unit (ALU), [21-27, 29](#)
- ASCII, [129, 131, 134, 138, 220](#)
- BASIC, [131, 182](#)
- Binary-coded decimal (BCD), [31, 61, 184](#)
- BIT, [75](#)
- Bits:
 - flag, [33](#)
 - least significant (LSB), [184](#)
 - manipulation, [32, 75](#)
 - most significant (MSB), [184](#)
 - start and stop, [139](#)
- Branching:
 - conditional, [80](#)
 - unconditional, [79](#)
- Buffering, [98](#)
 - address bus, [99](#)
 - data bus, [100](#)
- Bus(es), [22](#)
 - address, [29, 85, 98, 105, 110](#)
 - architecture, [24](#)
 - buffering, [98](#)
 - control, [100](#)
 - signals, [101](#)
 - testings, [105](#)
 - data, [22, 29, 85, 100, 116](#)
 - bi-directional, [22, 100, 105](#)
 - drivers, [93, 99-100](#)
 - testing, [105](#)
 - power, [98](#)
 - structures, [22](#)
 - voltage, [29](#)
- Bytes, [32](#)
- CALL, [82, 152](#)
- Capacitance, [14](#)
- Capacitors, [3, 5-6, 97](#)
 - bypass, [14](#)
 - charging time, [5](#)
 - filter, [2, 4, 14](#)
 - ripple factor of, [4](#)
 - input, [14](#)
 - sizing, [5](#)
 - time constants of, [9](#)
- Carry, [28](#)
 - flag, [51, 80](#)
- Cassettes, [121, 129, 145](#)
 - interface, [115, 145, 148-149](#)
 - Kansas City Standard, [146](#)
 - software, [146](#)
- CCF, [60](#)
- Central processors (see also Microprocessors), [21-27, 27](#)
 - architecture, [27](#)
 - control, [29, 32](#)
 - registers, [27-29](#)
 - status, [33](#)
 - synchronizing, [97](#)
 - testing, [127](#)
 - timing, [92](#)
- Characters, [213](#)
 - format, [214](#)
- Chip select, [116](#)
- Circuits:
 - complexity, [21, 23](#)
 - integrated, [10, 22](#)
 - layouts, [14](#)
 - protective, [10](#)
 - reset, [97](#)
- Clocks, [91, 209](#)
 - periods, [91](#)
 - real-time, [208](#)
 - single-stepping, [92, 105](#)
 - testing, [105](#)
- COM 8046, [220](#)
- COM 2017, [220](#)
- Communication, [138](#)
 - asynchronous, [139, 142](#)
 - parallel and serial, [138](#)
 - software, [148](#)
 - signal levels, [142](#)
 - standard, [144](#)
- Cooling, [17](#)
- Control section, [22](#)
- Controllers, intelligent, [181](#)
- Converters:
 - analog-to-digital, [164, 169](#)
 - analog to pulse width, [169](#)
 - binary-ramp counter, [161](#)
 - successive approximation, [164](#)
 - 3½-digit AC/DC, [162](#)
 - software, [205](#)
 - digital-to-analog, [184](#)
 - calibration, [188](#)
 - multiplying, [189](#)
 - R-2R, [184](#)
 - weighted-resistor, [184](#)
- Cost, [23](#)
- CP, [47](#)
- CPD, [48](#)
- CPDR, [48](#)
- CPL, [47](#)
- CPTR, [47](#)
- CPL, [60](#)
- CRT 8002, [213](#)
- CRT 8027, [213](#)
- Currents:
 - continuous, [6](#)
 - regulator, [5](#)
 - surge, [6](#)
- DAA, [61](#)
- Data, [22, 35, 112, 116](#)
 - acquisition, [198, 208](#)
 - ASCII, [128](#)
 - communication, [138](#)
 - formats, [22](#)
 - high- and low-order, [33](#)
 - rates, [142, 148, 220](#)
- DEC, [59, 65](#)
- Decoding:
 - hexadecimal, [135](#)
 - I/O, [91, 105-106, 208](#)
 - memory, [91, 105-106, 210](#)
 - testing, [111](#)
- Demultiplexers, [108, 206](#)

- DI, [62](#)
- Diodes, [3](#), [5-6](#), [97](#)
 - bridges, [5-6](#), [16](#)
 - silicon, [3](#)
 - zener, [6](#), [10](#)
- Direct memory access (DMA), [99](#), [129](#)
- Displays:
 - cathode-ray tube (CRT), [129](#), [136](#), [113](#)
 - hexadecimal, [134](#)
 - light-emitting diode (LED), [93](#), [121](#), [129](#), [134](#), [153](#)
 - octal, [134](#)
 - video, [121](#), [183](#), [213](#)
 - visual, [129](#), [134](#)
- DJNZ, [83](#)
- Drivers:
 - bus, [93](#)
 - display, [93](#)
 - LED, [93](#)
- EI, [62](#)
- 8060A, [24](#), [31](#), [91](#)
- 8212, [100](#)
- EX, [44](#)
- EXX, [44](#)
- Fanout, [96](#)
- Farads, [5](#)
- Flags, [33](#)
 - carry (C), [31](#), [80](#)
 - condition, [33-34](#)
 - status, [33](#)
 - zero (Z), [79](#), [80](#)
- Flip-flops, [92](#), [132](#)
- Frequency shift keying (FSK), [146](#)
- Full-wave bridges (see also Rectifiers), [3](#), [5](#)
- Fuses, [17](#)
- Grounds, [15](#)
 - buses, [15](#)
 - common, [14](#)
 - references, [11](#)
 - single-point, [15](#)
- HALT, [39](#), [62](#)
- Heat sinks, [16](#)
- HP7340, [125](#)
- IM, [62](#)
- IN, [85](#), [122](#)
- INC, [58](#), [64](#)
- IND, [87](#)
- INDR, [87](#)
- Inductance, [14](#)
- INI, [86](#)
- INIR, [86](#)
- Input, [21](#), [85](#), [122](#)
 - filters, [2-3](#)
- Input/output, [121](#), [129](#)
 - decoding, [91](#), [109](#)
 - testing, [111](#)
 - instructions, [32](#), [83](#)
 - ports, [98](#), [105](#), [108](#)
 - read, [106](#)
 - registers, [91](#)
 - request, [30](#), [106](#)
 - testing, [122](#), [127](#)
 - write, [106](#)
- Instructions, [21](#)
 - arithmetic and logical, [31](#)
 - 8-bit, [42](#)
 - general purpose, [60](#)
 - 16-bit, [64](#)
 - bit manipulation, [32](#), [75](#)
 - block transfer and search, [31](#), [44](#)
 - call and return, [32](#), [82](#), [152](#)
 - CPU control, [32](#), [60](#)
 - cycle, [91](#)
 - exchange, [28](#), [31](#), [44](#)
 - execution, [92](#)
 - fetch cycle, [29](#), [91-92](#)
 - formats, [32](#)
 - input and output, [32](#), [85](#), [88](#), [122](#)
 - jump, [32](#), [78](#)
 - load, [31](#)
 - 8-bit, [34](#)
 - 16-bit, [38](#)
 - pop, [43](#)
 - push, [42](#)
 - restart, [152](#)
 - rotate and shift, [31](#), [66](#)
 - sets, [33](#)
 - single-stepping, [92](#)
 - testing, [105](#)
 - types, [31](#)
- Interfaces:
 - cassette, [145](#)
 - tuning, [149](#)
 - clock, [209](#)
 - RS-232C, [213](#)
 - serial, [129](#), [138](#), [142](#)
 - 3V₁-digit AC/DC, [192](#)
 - testing, [205](#)
- Interrupts, [30](#), [62](#), [84](#)
 - non-maskable, [30](#), [84](#)
 - page address, [29](#)
- JP, [78](#)
- JR, [79](#)
- Karnaugh City Standard, [142](#)
- Keyboards, [118](#), [121](#), [129](#)
 - ASCII, [129](#), [134](#)
 - bounce, [132](#)
 - encoders, [131-132](#), [220](#)
 - hexadecimal, [133](#)
 - input software, [163](#)
- KR2376, [220](#)
- LD, [34](#)
- LDD, [46](#)
- LDDR, [46](#)
- LDI, [43](#)
- LDIR, [46](#)
- Light-emitting diodes (LED), [93](#), [121](#)
 - drivers, [93](#)
- Leads, [7](#), [29](#)
 - TTL, [24](#)
- Logic analyzers, [91](#), [93](#), [99](#)
- Low-power Schottky TTL (LSTTL), [98](#)
- Machine cycles, [28](#), [91](#)
- Memory, [21](#), [32](#), [91](#), [112](#)
 - addresses, [34](#), [97](#), [110](#)
 - banks, [110](#), [117](#)
 - contents, [34](#)
 - decoding, [91](#), [105](#), [110](#)
 - testing, [111](#)
 - direct memory access (DMA), [99](#)
 - display and replace, [151](#), [153](#)
 - dynamic, [116](#)
 - erasable-programmable read-only (EPROM), [112](#), [115](#), [152](#)
 - erasers, [177](#)
 - programmers, [173](#)
 - automatic, [174](#)
 - manual, [173](#)
 - locations, [78](#)
 - map, [117](#)
 - page, [113](#)
 - programmable, [27](#), [110](#)

random-access (RAM), [116](#)
 read, [30](#), [91](#), [106](#)
 cycles, [117](#)
 read-only (ROM), [110](#), [112](#), [173](#)
 character-generator, [213](#)
 diode-matrix, [113](#)
 programmable (PROM), [112](#)
 read/write (RWM), [112](#), [116](#)
 refresh, [29-30](#), [116](#)
 request, [30](#), [116](#)
 slow, [97](#)
 static, [116](#)
 storage, [112](#), [121](#), [145](#)
 testing, [127](#)
 write, [30](#), [91](#), [106](#)
 cycles, [117](#)
 Microcomputers, [21](#)
 construction, vii, [27](#), [91](#)
 definition of, [21](#)
 design of, [21](#), [27](#)
 single-board, [183](#)
 system, [22](#)
 Microprocessors (see also Central processors), [21](#)
 architecture, [21](#), [27](#)
 common, [24](#)
 definition of, [22](#)
 Z80, [24](#), [27](#)
 Monitors (see also Software), [113](#), [118](#), [134](#), [151](#), [173](#)
 cold start, [151](#)
 command recognition, [161](#)
 execute, [151](#), [152](#), [171](#)
 keyboard input, [163](#)
 memory display and replace, [151](#), [153](#), [168](#)
 register display and replace, [151](#), [154](#), [169](#)
 restart, [167](#)
 serial input/output, [151](#), [156-157](#), [159](#)
 UART diagnostic, [156](#)
 warm start, [151-152](#), [160](#)
 Multiplexers, [22](#), [117](#)
 NEG, [60](#)
 No operation (NOP), [30](#), [32](#), [61-62](#)
 Nyquist criterion, [197](#)
 Operands, [33](#)
 Operating systems, [151](#)
 Operation code, [29](#)
 OR, [34](#), [53](#)
 Oscilloscopes, [91](#), [93](#)
 OTDR, [90](#)
 OTIR, [89](#)
 OUT, [88](#), [122](#)
 OUTD, [89](#)
 OUTI, [88](#)
 Output, [22](#), [88](#), [122](#)
 Overflow, [28](#)
 Overvoltage protectors, [17](#)
 Parity, [28](#)
 Pascal, [183](#)
 Peak inverse voltages (PIV), [4](#)
 Peripherals, [121](#), [129](#), [151](#)
 synchronizing, [130](#)
 POP, [43](#)
 Ports, [33](#), [85](#), [98](#), [105](#), [108](#)
 hexadecimal output, [136](#)
 octal, [136](#)
 parallel and serial, [129](#), [183](#)
 Power dissipation, [4](#), [15](#)
 Power supplies, [1](#), [13](#)
 DC, [1](#)
 Printed-circuit boards, [21](#)
 Programs:
 debugging, [153](#)
 development, [153](#)
 PUSH, [42](#)
 Rectifiers (see also Full-wave bridges), [6](#), [14](#)
 bridge, [2](#), [3](#), [16](#)
 full-wave, [3](#), [5](#)
 silicon-controlled (SCR), [18-19](#)
 Refresh, [29-30](#), [116](#)
 Registers, [27-28](#)
 accumulator (A), [27-28](#), [33](#)
 contents, [34](#)
 display and replace, [151](#), [154](#)
 8-bit (B, C, D, E, [H](#), [L](#)), [27](#), [117](#)
 flag (F), [27-28](#), [33](#)
 general purpose, [28](#)
 index (IX, IY), [29](#)
 instruction, [29](#)
 interrupt page address (I), [29](#)
 main and alternate, [28-29](#)
 memory refresh (R), [29](#)
 pairs, [28](#), [33](#), [39](#)
 program counter (PC), [28](#), [32](#), [78](#), [82](#), [152](#)
 sets, [27-28](#)
 16-bit (BC, DE, HL), [27](#)
 special purpose, [28](#)
 stack pointers (SP), [28](#), [42](#), [152](#)
 Regulators, voltage (see Voltages, regulators)
 Requests, [106](#)
 input/output, [106](#)
 memory, [106](#)
 read, [106](#)
 write, [106](#)
 RES, [78](#)
 Resets, [62](#), [97](#), [152](#)
 automatic, [97](#)
 manual, [97](#)
 testing, [105](#), [127](#)
 Resistance, [3](#), [6](#), [15](#)
 series, [6](#), [8](#)
 thermal, [16](#)
 Resistors, [19](#), [183](#)
 ladder, [183](#)
 variable, [8](#)
 Resolution, [184](#), [187](#), [198](#)
 RET, [83](#)
 RETI, [84](#)
 RETN, [84](#)
 Ripple factor, [4](#)
 RL, [68](#)
 RLA, [66](#)
 RLC, [67](#)
 RLCA, [66](#)
 RLD, [74](#)
 RR, [70](#)
 RRA, [68](#)
 RRC, [68](#)
 RRCA, [66](#)
 RRD, [75](#)
 RS-232C, [144](#), [213](#)
 RST, [85](#), [152](#)
 Sample rates, [194](#), [197](#)
 SBC, [53](#), [64](#)
 SCF, [60](#)
 SET, [76](#)
 78H05, [10](#), [16](#)
 7812, [12](#)
 7912, [12](#)
 Short-circuits, [18](#)
 Sign, [28](#)
 Sine waves, [3](#)

- 6800, [24](#)
- 6502, [24](#)
- SLA, [71](#)
- Software (see also Monitors), [24](#)
 - monitor, [151](#)
 - single-stepping, [92](#)
- SRA, [72](#)
- SRL, [73](#)
- Stacks, [28](#), [32](#), [43](#), [82](#), [163](#)
- Strobes:
 - data-ready, [130](#)
 - duration, [132](#)
 - key-pressed, [139](#)
- SUB, [52](#)
- Subroutines, [28](#), [82](#), [118](#)
- Surge currents, [6](#)
- Terminals, [213](#)
- Testing:
 - dynamic, [127](#)
 - static, [123](#)
- Thermal considerations, [15](#)
- Timers, [130](#)
- Transformers, [1](#), [6](#)
 - primary input to, [3](#)
 - secondary output from, [3-4](#)
- Transistor-transistor logic (TTL), [93](#), [98](#), [217](#)
 - levels, [147](#)
 - loads, [93](#)
 - low-power Schottky (LSTTL), [98](#), [217](#)
 - outputs, [138](#), [146](#)
- Transistors, [8](#), [12](#)
 - FAMOS, [115](#), [173](#)
 - series-pass, [10](#)
 - wide-band, [14](#)
- 2114, [117](#)
- 2102A, [112](#)
- 2708, [113](#), [173](#)
- 2716, [113](#), [173](#)
- Universal asynchronous receiver/transmitter (UART), [130](#), [220](#)
 - diagnostic, [156](#)
 - output, [146](#)
 - pinout, [139](#)
- Voltages:
 - alternating current, [1](#)
 - comparators, [7-8](#)
 - control element, [7](#)
 - direct current (DC), [1](#)
 - drops, [3](#), [6](#), [11](#), [14](#)
 - input and output, [7](#), [14](#)
 - loads, [5](#)
 - peak, [4](#), [15](#)
 - peak inverse (PIV), [6](#)
 - reference, [7](#), [10](#)
 - regulators, [1](#), [3-4](#), [7](#), [10](#), [16](#)
 - choosing, [10](#)
 - overloads, [10](#)
 - series, [8](#)
 - three-terminal, [9-10](#)
 - ripple, [4-5](#), [14](#)
 - root mean square (RMS), [3](#), [6](#)
 - sine waves, [2](#)
 - transients, [6](#)
 - translators, [7-8](#)
 - VAC, [1](#), [3](#)
 - waveforms, [3-4](#)
- Voltmeters, [93](#), [184](#), [199](#)
- Waits, [30](#), [92](#)
- XOR, [56](#)
- Z80, [24](#), [27](#)
 - bus structure, [25](#)
 - busing and control logic, [91](#)
 - pinout, [29](#)
- Z80 Applications Processor (ZAP), vii, [1](#), [91](#)
 - testing, [123](#), [127](#)
- Zero, [28](#)
 - flag, [75](#), [80](#)

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

BY THE SAME AUTHOR

CIARCIA'S CIRCUIT CELLAR

1979, 128 pages

This volume provides a wealth of imaginative and practical microcomputer projects. Selected from the popular series in *BYTE* magazine, topics included D/A conversion, programming EPROMS, AC remote-controlled appliances, digitized speech, and touch input video display.

CIARCIA'S CIRCUIT CELLAR, VOLUME II

1981, 224 pages

Presented in the same easy-going style and sprinkled with amusing anecdotes, this second volume offers more practical uses for the home computer. Focused on how microcomputers can be uniquely interfaced to our environment, projects cover building a computer-controlled home security system, computerizing appliances, transmitting digital information over a beam of light, building the Intel 8086 microprocessor system design kit, and input/output expansion for the TRS-80.

ALSO FROM BYTE BOOKS

THREADED INTERPRETIVE LANGUAGES

Ronald Loeliger, Senior Analyst with Intermetrics, Inc.

1981, 272 pages

This text on threaded languages (such as FORTH) develops an interactive, extensible language with specific routines for the Zilog Z80 microprocessor.

BEGINNER'S GUIDE FOR THE UCSD PASCAL SYSTEM

Kenneth J. Bowles, Director of the Institute for Information Systems, University of California, San Diego

1980, 204 pages

Written by the originator of the UCSD Pascal System for users of microcomputers and minicomputers, this book is both an orientation guide to the System and an invaluable reference tool for creating advanced applications.

THE BYTE BOOK OF PASCAL

Blaise W. Lillick, Editor

1980, 334 pages

Written for both potential and experienced computer users, this valuable software resource introduces the Pascal language and examines its merits and possible implementations.

Build Your Own Z80 Computer: Design Guidelines and Application Notes

"There is a major need for a book such as this. The information is not readily available elsewhere. Or anywhere. There are dozens (hundreds?) of microprocessor books, but nearly all deal with software and treat hardware as abstractions or block diagrams. Ciarcia's book is literally filled with very useful and practical "hands-on" hardware advice, tips and techniques... The book will do for the reader what no other microprocessor book or manufacturer's literature I know of does: It will enable a person to actually buy individual parts and assemble them into a working microcomputer—with peripherals and options! That's very important. Too bad we couldn't have had such a book years ago."

—Forrest Hims, III
Contributing Editor of **POPULAR ELECTRONICS**

"To my knowledge the material covered in this book is not available elsewhere. There is sufficient detail to enable an individual with previous experience to assemble a working Z80-based microcomputer from the component level. The design trade-offs, the circuits, the software, and the test circuits and procedures are discussed at a level sufficient for the book to have educational value even if one did not actually construct a Z80-based system."

—Joseph Nichols
Digital Analysis Corporation

About the Author

Steve is a computer consultant, electrical engineer, author of *BYTE* magazine's most popular column, "Ciarcia's Circuit Cellar," and a "national technological treasure."
